
Mif2Go User's Guide

for **Mif2Go** Version 4.1, Update 55

May 15, 2013

Omni Systems, Inc.

Overview

Lists

§ Contents	1
§ Figures	31
§ Tables	35

How to set up and use Mif2Go

§ About this guide.	41
§1 Getting started with Mif2Go	51
§2 Planning a conversion project	65
§3 Converting a book or document.	77
§4 Editing configuration files	91
§5 Setting basic conversion options	109

Configuring print RTF output

§6 Converting to print RTF.	141
----------------------------------	-----

Configuring on-line Help output

§7 Producing on-line Help	199
§8 Generating WinHelp	243
§9 Generating Microsoft HTML Help	295
§10 Generating OmniHelp	341
§11 Generating JavaHelp or Oracle Help	373
§12 Generating Eclipse Help	403

Configuring HTML/XML output

§13 Converting to HTML/XHTML	423
§14 Converting to generic XML	457
§15 Converting to DITA XML	473
§16 Configuring DITA maps	539
§17 Converting to DocBook XML	557
§18 Splitting and extracting files	585
§19 Creating HTML links.	609
§20 Providing navigation in HTML	627
§21 Mapping text formats to HTML/XML	645
§22 Setting up CSS for HTML	681
§23 Including graphics in HTML	703
§24 Converting tables to HTML	727

Web Accessibility Initiative

§25 Generating WAI markup for HTML	755
§26 Identifying HTML table structure for WAI	763
§27 Marking HTML table cells for WAI	775

Advanced topics

§28 Working with macros	787
§29 Working with FrameMaker markers	831
§30 Working with templates	849
§31 Working with graphics	869
§32 Working with content models	905
§33 Overriding configuration settings	919

Project workflow

§34 Automating Mif2Go conversions	933
§35 Producing deliverable results	955
§36 Converting via runfm	979
§37 Converting via DCL	995
§38 Generating intermediate output	1005

Reference

§A WAI marker library for HTML	1013
§B Distribution files	1017
§C Document and conversion files	1019
§D Technical support for Mif2Go	1029
§E DITA <bookmeta> template	1039
§F Content model configuration	1043

Indexes

§ RTF keyword index	1047
§ HTML/XML keyword index	1059
§ Subject index	1085

Figures	31
Tables	35
About this guide	41
1 Getting started with Mif2Go	51
1.1 What you need to know	51
1.1.1 How Mif2Go is organized	51
1.1.2 File, directory, and path names	51
1.1.3 Output types you can specify	52
1.1.4 Languages and character sets	53
1.2 What you need to have	53
1.3 What you need to do	54
1.3.1 Set up a framework for Omni Systems applications	54
1.3.2 Download a Mif2Go distribution	56
1.3.3 Install Mif2Go	56
1.3.4 Make Omni Systems executables accessible	57
1.3.5 Obtain tools for Help systems or eBooks	58
1.3.6 Establish system-wide configuration settings	58
1.3.7 Locate document-specific settings	60
1.3.8 Obtain a file comparison tool (optional)	60
1.3.9 Download the Mif2Go User's Guide (optional)	61
1.4 How to update Mif2Go	61
1.4.1 Change from the evaluation version to a licensed version	61
1.4.2 Update your Mif2Go installation	61
1.4.3 Try out Mif2Go beta executables	62
1.5 How Mif2Go works	62
1.6 How to start and stop Mif2Go	63
1.7 How to work with Mif2Go	63
1.8 How to uninstall Mif2Go	64
2 Planning a conversion project	65
2.1 Naming files, directories, and paths	65
2.2 Naming FrameMaker formats	66
2.3 Understanding Mif2Go configuration files	66
2.4 Importing formats from a conversion template	67
2.5 Preparing documents for conversion	69
2.5.1 Updating your document in FrameMaker	69
2.5.2 Planning for graphics processing	69
2.5.3 Replacing embedded graphics with referenced graphics	69
2.5.4 Setting up cross references to and from text insets	70
2.5.5 Creating hotspots for hypertext links	72
2.5.6 Producing a single output file from a FrameMaker book	73
2.5.7 Preparing a structured document for conversion	73
2.6 Establishing a conversion environment	74

2.7	Setting up multiple interlinked HTML projects	75
2.8	Preparing deliverables after conversion	75
3	Converting a book or document	77
3.1	Checking set-up and conversion requirements	77
3.2	Starting Mif2Go	77
3.3	Creating a Mif2Go conversion project	78
3.4	Choosing project set-up options	79
3.4.1	Importing formats from a FrameMaker template	79
3.4.2	Converting FrameMaker system variables to text	80
3.4.3	Generating and updating your document	81
3.4.4	Including FrameMaker-generated files	81
3.4.5	Understanding configuration settings for general set-up options	81
3.4.6	Choosing output-specific set-up options	82
3.5	Understanding how Mif2Go sets up a project	82
3.6	Converting documents	82
3.7	Choosing final conversion options	83
3.7.1	Understanding how export options work	84
3.7.2	Specifying output type and file extension	84
3.7.3	Choosing input source and disposition	85
3.7.4	Figuring out graphics export options	85
3.7.5	Choosing postprocessing options	88
4	Editing configuration files	91
4.1	Working with Mif2Go configuration files	91
4.2	Editing files with the Configuration Manager	91
4.2.1	Understanding how to use the Configuration Manager	92
4.2.2	Starting the Configuration Manager	93
4.2.3	Setting Configuration Manager preferences	93
4.2.4	Establishing a starting point	95
4.2.5	Choosing a configuration category or file type	95
4.2.6	Understanding variable vs. fixed names and keys	95
4.2.7	Choosing the kind of change to make	96
4.2.8	Selecting a configuration section	100
4.2.9	Selecting a configuration setting	100
4.2.10	Selecting a configuration file	101
4.2.11	Specifying a final value	101
4.3	Understanding where project settings come from	102
4.4	Understanding the rules for configuration settings	102
4.5	Specifying file paths in configuration settings	105
4.6	Using wildcards in configuration settings	106
4.7	Commenting out configuration sections	107
4.8	Ending a configuration file	107
5	Setting basic conversion options	109
5.1	Specifying operating settings	109
5.1.1	Checking output type and file extension	110
5.1.2	Excluding files from book conversions	110
5.1.3	Reusing or discarding MIF files	111

5.1.4 Reusing or discarding ASCII DCL files	111
5.1.5 Checking for broken links in HTML or XML output	112
5.1.6 Skipping the Mif2Go <i>Export</i> and <i>Finished</i> dialogs.	112
5.1.7 Specifying how to treat cases, spaces, and wildcards	113
5.1.8 Reordering text flows	113
5.1.9 Converting system variables to text	114
5.1.10 Preserving Word-generated cross-reference markers	114
5.2 Logging conversion events.	115
5.3 Identifying files and objects	117
5.3.1 Understanding how Mif2Go creates identifiers	117
5.3.2 Working with FrameMaker ObjectIDs	118
5.3.3 Working with FrameMaker cross-reference IDs.	119
5.3.4 Working with Mif2Go FileIDs	119
5.4 Applying FrameMaker conditions and variables	122
5.4.1 Applying condition Show/Hide settings	123
5.4.2 Replacing values of FrameMaker user variables.	123
5.5 Converting FrameMaker-generated files	124
5.5.1 Converting FrameMaker TOC and IX files.	124
5.5.2 Preventing conversion of other generated files	125
5.5.3 Activating hypertext links in a converted index	125
5.5.4 Making <i>See</i> and <i>See also</i> index entries into useful links.	125
5.6 Generating/updating before converting	126
5.7 Processing graphics	126
5.7.1 Understanding which graphics are included	126
5.7.2 Choosing how to convert graphics	127
5.7.3 Choosing when to convert graphics	131
5.7.4 Identifying exported graphics files	133
5.8 Converting structured documents.	135
5.9 Converting equations	136
5.9.1 Understanding how equations are processed.	136
5.9.2 Specifying equation size and DPI	136
5.9.3 Specifying equation output format	137
5.9.4 Providing a file-name suffix for equations	137
5.9.5 Positioning equations in RTF output.	137
5.10 Creating hotspots for hypertext links	138
5.10.1 Delimiting a hotspot with a character format	138
5.10.2 Making an entire paragraph into a hotspot	138
5.10.3 Delimiting a hotspot with a color	139
5.11 Repurposing FrameMaker markers	139
6 Converting to print RTF	141
6.1 Converting to Word: a one-way street	141
6.1.1 Understanding differences in implementation.	142
6.1.2 Understanding differences in file sizes	143
6.1.3 Understanding why round-tripping is not an option	143
6.1.4 Migrating a document from FrameMaker to Word.	144
6.1.5 Developing a workflow using Word for reviews	144
6.2 Setting up a print RTF project	145
6.2.1 Creating a print RTF project	145
6.2.2 Choosing set-up options for a print RTF project.	146

6.2.3	Specifying output file extension	147
6.2.4	Specifying the default output language and code page	147
6.2.5	Constraining the number of bookmarks in Word.	148
6.2.6	Importing a Word template	148
6.3	Adjusting output for different versions of Word	149
6.4	Converting a FrameMaker book to print RTF	150
6.5	Specifying document layout options.	151
6.5.1	Understanding page layout restrictions	151
6.5.2	Eliminating large top or bottom margins	151
6.5.3	Using text frames to solve spacing problems.	152
6.5.4	Maintaining pagination in Word	152
6.5.5	Managing page and section breaks	152
6.5.6	Specifying columns and gaps.	153
6.5.7	Adjusting sidehead width for Word	153
6.5.8	Converting footnotes	153
6.5.9	Converting headers and footers	154
6.5.10	Converting special text flows for RTF output	156
6.5.11	Handling different page size or orientation	157
6.6	Converting system variables to text for RTF	157
6.7	Converting paragraph and character formats	158
6.7.1	Mapping paragraph formats to RTF styles	158
6.7.2	Merging paragraph formats	159
6.7.3	Converting sidehead formats	159
6.7.4	Converting run-in headings	160
6.7.5	Converting autonumbered formats.	160
6.7.6	Converting bulleted formats.	162
6.7.7	Converting reference frames for Word	162
6.7.8	Converting character formats.	163
6.7.9	Removing unused formats	163
6.8	Converting tabs and spaces.	163
6.8.1	Understanding differences in tab behavior	163
6.8.2	Understanding differences in spaces	164
6.8.3	Altering tab behavior for Word output	164
6.8.4	Altering font metrics to adjust tabs	165
6.9	Specifying font usage	166
6.9.1	Setting default font parameters	166
6.9.2	Remapping fonts	166
6.9.3	Specifying font types	167
6.9.4	Specifying font encoding for non-Western characters.	168
6.9.5	Specifying font encoding for FrameMaker 8 Unicode	169
6.9.6	Removing unused fonts	170
6.10	Modifying text appearance	170
6.10.1	Adjusting line spacing	170
6.10.2	Adjusting paragraph spacing	170
6.10.3	Adjusting small caps	172
6.10.4	Specifying a style for quotes	172
6.10.5	Mapping high ASCII characters for RTF output	172
6.10.6	Specifying text color	172
6.10.7	Hiding white text	173
6.10.8	Hiding content in Word	173

6.10.9	Omitting content from RTF output	174
6.10.10	Replacing content in RTF output	174
6.11	Converting cross references and hypertext links	174
6.11.1	Including ObjectIDs for Word links and cross references	175
6.11.2	Converting cross references to Word	175
6.11.3	Converting hypertext links to Word	178
6.11.4	Locking hypertext links to allow revision tracking	178
6.11.5	Enabling interfile cross references and hypertext links	179
6.11.6	Replacing building blocks in master-page references	181
6.12	Converting generated files to print RTF	181
6.12.1	Specifying which generated files to convert	182
6.12.2	Activating links in converted index and list files	182
6.12.3	Making the entire text of each list entry an active link	182
6.12.4	Ensuring link targets are present in RTF output	183
6.12.5	Correcting <\$nopage> index links	184
6.13	Converting tables to print RTF	184
6.14	Managing graphics for print RTF	186
6.14.1	Understanding graphics requirements for Word	186
6.14.2	Converting referenced graphics	187
6.14.3	Converting embedded graphics	189
6.14.4	Limiting bitmap resolution and color depth	190
6.14.5	Managing callouts added to graphics	190
6.14.6	Positioning graphics and wrapping text	191
6.14.7	Preserving graphics scale in Word	191
6.14.8	Accommodating graphics in multiple versions of Word	192
6.14.9	Including file names of referenced graphics in Word	192
6.14.10	Linking instead of embedding referenced graphics	193
6.14.11	Embedding graphics in converted RTF files	193
6.14.12	Updating fields in Word to show linked graphics	193
6.15	Including RTF code for Word output	194
6.16	Turning on revision tracking in Word	194
6.17	Managing Word output after conversion	195
6.17.1	Supporting more than one version of Word	195
6.17.2	Including index terms in Word	195
6.17.3	Producing ASCII text from a converted Word document	196
6.17.4	Combining RTF files into a Word master document	197
6.17.5	Checking print RTF output files for Mif2Go version	197
6.18	Converting to OpenOffice or StarOffice	197
7	Producing on-line Help	199
7.1	Weighing Help-system alternatives	199
7.1.1	Considering Help-system features	200
7.1.2	Understanding the effects of mid-topic links	200
7.1.3	Evaluating Microsoft Windows Help (WinHelp)	200
7.1.4	Evaluating Microsoft HTML Help	201
7.1.5	Evaluating WebHelp	201
7.1.6	Evaluating OmniHelp	202
7.1.7	Evaluating JavaHelp and Oracle Help for Java	202
7.1.8	Evaluating Eclipse Help	202
7.2	Setting up a Help system project	203

7.2.1	Checking automatic Help topic assignments	203
7.2.2	Configuring run-in paragraphs	203
7.2.3	Specifying additional processing after conversion	203
7.2.4	Compiling and distributing Help systems	204
7.3	Producing contents and index for Help systems	204
7.3.1	Understanding how Mif2Go produces contents and index	205
7.3.2	Including FrameMaker TOC and IX in Help systems	205
7.3.3	Grouping contents entries	206
7.3.4	Modifying contents or index production for HTML-based Help	206
7.3.5	Modifying contents or index production for WinHelp	208
7.4	Configuring contents entries for Help systems	209
7.4.1	Understanding how contents levels are assigned	209
7.4.2	Setting contents levels for WinHelp	209
7.4.3	Including contents entries in HTML-based Help	209
7.4.4	Setting contents levels for HTML-based Help	210
7.5	Configuring index entries for Help systems	211
7.5.1	Understanding how Mif2Go creates Help index entries	211
7.5.2	Preparing index entries for Microsoft Help Viewer	211
7.5.3	Limiting length of index entries for HTML Help or WinHelp	212
7.5.4	Omitting intermediate index-range entries	212
7.5.5	Treating commas as potential index level separators	213
7.5.6	Combining index levels for HTML-based Help	213
7.5.7	Configuring <i>See</i> and <i>See also</i> entries for HTML-based Help	214
7.5.8	Specifying index link destinations for HTML-based Help	215
7.5.9	Customizing index sort order	216
7.6	Providing related-topic links for Help systems	219
7.6.1	Understanding related-topic links	219
7.6.2	Understanding how ALinks work	220
7.6.3	Understanding how KLinks work	221
7.6.4	Adding related-topic link keywords in FrameMaker	221
7.6.5	Adding ALink and KLink jumps in FrameMaker	222
7.6.6	Creating target-and-jump ALinks for HTML-based Help	224
7.6.7	Specifying ALink and KLink list-link destinations	224
7.7	Jumping to secondary windows in Help systems	224
7.7.1	Assigning secondary windows for WinHelp	224
7.7.2	Assigning secondary windows for HTML-based Help	225
7.8	Creating pop-up topics for Help systems	225
7.8.1	Understanding pop-up hotspots, links, and topics	225
7.8.2	Defining a pop-up hotspot	226
7.8.3	Displaying a topic in a pop-up window	226
7.9	Including expandable sections in Help topics	226
7.9.1	Understanding Mif2Go expandable drop-down sections	227
7.9.2	Setting up expandable sections for your document	227
7.9.3	Delimiting expandable drop-down sections	228
7.9.4	Configuring drop-down links	230
7.9.5	Configuring drop-down blocks	233
7.9.6	Providing CSS for drop-down links and blocks	233
7.9.7	Deploying JavaScript code for drop-down sections	234
7.9.8	Emulating Web Works Publisher drop-down hotspots	237
7.10	Setting up Context Sensitive Help (CSH)	239

7.10.1 Understanding how CSH works	240
7.10.2 Specifying CSH mappings	241
7.11 Setting up a dynamic modular Help system	241
8 Generating WinHelp	243
8.1 Obtaining tools for WinHelp	243
8.2 Setting up a WinHelp project	243
8.2.1 Setting up a WinHelp project	244
8.2.2 Choosing set-up options for a WinHelp project	244
8.2.3 Deciding where to locate configuration settings	245
8.2.4 Preparing a document for conversion to WinHelp	246
8.2.5 Understanding initial set-up requirements	246
8.2.6 Deciding whether to regenerate the WinHelp project file	246
8.2.7 Accommodating platform differences	247
8.2.8 Setting basic WinHelp options in the configuration file	248
8.2.9 Including ObjectIDs in WinHelp	249
8.2.10 Handling page breaks and section breaks	249
8.2.11 Providing multiple .hlp files	249
8.2.12 Integrating WinHelp from RoboHelp	250
8.2.13 Compiling a WinHelp project	250
8.2.14 Checking WinHelp RTF files for Mif2Go version	251
8.3 Converting text	252
8.3.1 Converting formats for WinHelp	252
8.3.2 Converting special characters	254
8.3.3 Removing unused formats and fonts	257
8.3.4 Converting autonumbers	257
8.3.5 Replacing paragraph or character content	257
8.3.6 Specifying text color	258
8.3.7 Converting footnotes	258
8.4 Converting cross references	259
8.4.1 Creating help context markers	259
8.4.2 Specifying cross-reference destination files	259
8.4.3 Specifying cross-reference jump destinations	260
8.4.4 Specifying WinHelp options for cross-reference formats	260
8.4.5 Limiting cross-reference text	261
8.5 Converting tables to WinHelp RTF	261
8.5.1 Positioning tables and table titles	261
8.5.2 Adjusting table appearance	261
8.5.3 Converting table rows to topics and table cells to pop-ups	262
8.6 Managing graphics for WinHelp	263
8.6.1 Choosing a graphics format for WinHelp	263
8.6.2 Avoiding the GDI resource leak	264
8.6.3 Positioning graphics in WinHelp	264
8.6.4 Displaying graphics in pop-ups for WinHelp	265
8.7 Converting generated files for WinHelp	265
8.7.1 Converting lists of paragraph references	266
8.7.2 Converting indexes and lists of marker references	266
8.8 Configuring WinHelp topics	267
8.8.1 Creating WinHelp topics	267
8.8.2 Assigning properties to formats for topics and hotspots	268
8.8.3 Configuring topic titles for WinHelp	271

8.9	Creating jumps and pop-ups for WinHelp	272
8.9.1	Identifying WinHelp jump destinations with FileIDs	273
8.9.2	Configuring pop-up topics	273
8.9.3	Creating hotspots for jumps and pop-ups in WinHelp	274
8.9.4	Using cross references for jumps and pop-ups	276
8.9.5	Using hypertext links for jumps and pop-ups	276
8.9.6	Disallowing hypertext links for jumps and pop-ups	277
8.9.7	Specifying jumps to secondary windows in WinHelp	277
8.9.8	Specifying jumps to external files	278
8.9.9	Using the same content for both normal topics and pop-ups	278
8.9.10	Creating a glossary pop-up: an example	280
8.9.11	Configuring alternative jumps and pop-ups	280
8.9.12	Specifying the scope of alternative jumps and pop-ups	283
8.10	Invoking WinHelp macros	284
8.10.1	Using a hypertext marker to invoke a macro	284
8.10.2	Assigning a hotspot property to invoke a macro	284
8.11	Creating related-topic links in WinHelp	285
8.11.1	Understanding KLink limitations	285
8.11.2	Adding ALinks and KLinks with markers	285
8.11.3	Adding related-topic keywords with formats	285
8.11.4	Inserting WinHelp macros for ALink jumps	286
8.12	Configuring index entries for WinHelp	287
8.12.1	Designating index level separators	287
8.12.2	Eliminating duplicate keywords	287
8.12.3	Keeping or discarding "See also" entries	288
8.12.4	Using FrameMaker Index markers	288
8.13	Configuring contents for WinHelp	288
8.13.1	Naming and configuring Help files and titles	288
8.13.2	Specifying heading formats and levels for contents	289
8.13.3	Assembling WinHelp contents from the command line	291
8.14	Creating browse sequences	292
8.14.1	Setting up an automatic browse sequence	292
8.14.2	Specifying browse numbers	292
8.14.3	Setting up multi-file browse sequences	293
8.14.4	Setting up branching browse sequences	293
9	Generating Microsoft HTML Help	295
9.1	Understanding how Mif2Go produces HTML Help	295
9.2	Understanding why Unicode is not the answer	296
9.3	Setting up an HTML Help project	297
9.3.1	Creating an HTML Help project	297
9.3.2	Choosing set-up options for an MS HTML Help project	298
9.3.3	Deciding where to locate configuration settings	299
9.3.4	Organizing source files for HTML Help	299
9.3.5	Specifying a project title for HTML Help	300
9.3.6	Deciding whether to compile HTML Help	300
9.3.7	Naming project and compiled files for HTML Help	300
9.3.8	Specifying a starting topic file for HTML Help	301
9.3.9	Regenerating the HTML Help project file	301
9.3.10	Locating graphics files for HTML Help	302
9.4	Customizing HTML Help display features	302

9.4.1	Using CSS and font tags with HTML Help.	303
9.4.2	Eliminating graphic and table indents from HTML Help.	303
9.4.3	Adding tabs and toolbar buttons to HTML Help.	303
9.4.4	Adding expandable sections to HTML Help.	305
9.5	Creating pop-ups for HTML Help.	305
9.5.1	Using HTML Help for pop-ups.	306
9.5.2	Using KeyHelp for pop-ups.	306
9.5.3	Using WinHelp for pop-ups.	307
9.6	Creating links and hypertext jumps in HTML Help.	307
9.6.1	Creating hypertext jumps to other CHM files.	307
9.6.2	Specifying href link syntax for HTML Help.	308
9.6.3	Linking to external files from compiled HTML Help.	308
9.7	Creating related-topic links for HTML Help.	309
9.7.1	Adding ALink keywords for HTML Help.	309
9.7.2	Adding ALink and KLink jumps for HTML Help.	309
9.7.3	Configuring ALink and KLink jumps for HTML Help.	310
9.7.4	Rolling your own macros for ALink jumps in HTML Help.	312
9.7.5	Using the same format or marker for ALink keywords and jumps.	312
9.7.6	Creating buttons for other types of related-topic links.	317
9.8	Using secondary windows in HTML Help.	317
9.8.1	Defining secondary windows for HTML Help.	317
9.8.2	Jumping from a topic to a secondary window.	318
9.8.3	Jumping from contents or index to a secondary window.	318
9.9	Generating contents and index for HTML Help.	319
9.9.1	Choosing how to generate HTML Help contents and index.	319
9.9.2	Choosing whether to generate binary contents or index.	320
9.9.3	Generating contents and index with HTML Help Workshop.	321
9.9.4	Generating contents and index with Mif2Go	321
9.9.5	Configuring contents entries for HTML Help.	322
9.9.6	Providing mid-topic contents links in HTML Help.	323
9.9.7	Making the TOC track index links in HTML Help.	323
9.9.8	Customizing contents and index for HTML Help.	324
9.10	Converting generated files for HTML Help.	325
9.10.1	Converting lists of paragraph references.	325
9.10.2	Converting lists of marker references.	325
9.11	Providing full-text search (FTS) for HTML Help.	326
9.12	Setting up CSH for HTML Help.	326
9.12.1	Inserting CSH destinations in your document.	327
9.12.2	Determining whether you need map and alias files.	328
9.12.3	Specifying and generating a map file for CSH links.	329
9.12.4	Creating an alias file for CSH links.	330
9.12.5	Understanding alias-file entries.	330
9.12.6	Producing a list of aliases and associated topic titles.	331
9.13	Generating HTML Help in non-Western languages.	331
9.13.1	Converting from Unicode to Windows code pages.	331
9.13.2	Specifying locale and language for HTML Help.	332
9.13.3	Preventing inclusion of Unicode numeric references.	333
9.13.4	Coping with FrameMaker index-entry conversion defects.	333
9.14	Compiling and testing HTML Help.	333
9.14.1	Directing Mif2Go to run the HTML Help compiler.	333

9.14.2	Copying output files and compiling later	334
9.14.3	Compiling in a different language	335
9.14.4	Testing HTML Help generation.	335
9.14.5	Registering your HTML Help system for network use	335
9.15	Mapping and merging CHM files	336
9.15.1	Interlinking multiple CHM files	336
9.15.2	Synchronizing TOC references to slave CHM files.	338
9.15.3	Putting up with a binary index for merged CHM files	338
9.15.4	Merging CHM files	339
9.15.5	Comparing HHW settings for stand-alone vs. merged CHMs.	339

10 Generating OmniHelp 341

10.1	Understanding how OmniHelp works	341
10.2	Setting up OmniHelp viewer control files	342
10.2.1	Choosing XHTML vs. HTML OmniHelp control files	342
10.2.2	Making OmniHelp viewer control files available	343
10.2.3	Customizing OmniHelp viewer control files	343
10.2.4	Examining generated control and data files.	344
10.3	Setting up an OmniHelp project	345
10.3.1	Creating an OmniHelp project	345
10.3.2	Choosing set-up options for an OmniHelp project	346
10.3.3	Deciding where to locate configuration settings	347
10.3.4	Naming your OmniHelp project	347
10.3.5	Giving your OmniHelp project a title	348
10.3.6	Specifying the starting topic	348
10.3.7	Specifying memory requirements	348
10.3.8	Removing paths from interfile links for OmniHelp.	349
10.3.9	Getting OmniHelp supporting files in the right place	349
10.4	Using CSS with OmniHelp.	350
10.4.1	Specifying CSS for topics in OmniHelp	350
10.4.2	Understanding how CSS works in OmniHelp topics.	351
10.4.3	Specifying CSS for OmniHelp navigation frames.	352
10.5	Customizing OmniHelp display features	352
10.5.1	Configuring OmniHelp window usage and frameset dimensions	352
10.5.2	Altering OmniHelp top navigation frame content	353
10.5.3	Modifying OmniHelp navigation aids	353
10.5.4	Choosing whether to use cookies for OmniHelp	354
10.5.5	Localizing the OmniHelp interface	354
10.5.6	Modifying OmniHelp CSS classes	355
10.5.7	Modifying the OmniHelp template	356
10.6	Choosing navigation features for OmniHelp	356
10.7	Configuring contents and index for OmniHelp	357
10.7.1	Understanding OmniHelp contents and index creation	357
10.7.2	Choosing whether to use expanding contents or index	357
10.7.3	Choosing how far to expand contents and index subentries	358
10.7.4	Providing alternate expansion icons for contents or index	358
10.7.5	Excluding <i>Open All</i> and <i>Close All</i> buttons.	359
10.7.6	Redirecting <i>See</i> and <i>See also</i> index entries	359
10.8	Providing related-topic links in OmniHelp.	359
10.9	Jumping to secondary windows in OmniHelp	360

10.10	Configuring full-text search for OmniHelp	361
10.10.1	Understanding how OmniHelp FTS works	361
10.10.2	Generating search data	361
10.10.3	Making compound terms searchable	362
10.10.4	Supporting search for non-ANSI text	362
10.10.5	Specifying length of search terms	363
10.10.6	Excluding search terms	363
10.10.7	Excluding content from being searched	363
10.10.8	Using regular expressions in search	363
10.10.9	Highlighting search terms found in topics	364
10.11	Setting up CSH for OmniHelp	364
10.11.1	Specifying alias prefixes for OmniHelp CSH calls	364
10.11.2	Referencing OmniHelp topic IDs from an application	365
10.11.3	Using redirect pages for OmniHelp CSH calls	365
10.11.4	Executing browser commands for OmniHelp CSH calls	366
10.12	Merging OmniHelp projects	366
10.12.1	Understanding the OmniHelp merge process	366
10.12.2	Listing and mapping OmniHelp subprojects	367
10.12.3	Providing TOC placeholders for OmniHelp subprojects	368
10.12.4	Deciding when to merge OmniHelp subprojects	369
10.13	Assembling OmniHelp files for viewing	369
10.14	Deploying OmniHelp	370
10.14.1	Starting with the default topic or a specified topic	371
10.14.2	Restarting where you left off	371
10.14.3	Coping with browser quirks	371
11	Generating JavaHelp or Oracle Help	373
11.1	Deciding which Java Help system to use	373
11.2	Obtaining tools for a Java-based Help system	373
11.3	Setting up a JavaHelp or Oracle Help project	374
11.3.1	Creating a JavaHelp or Oracle Help for Java project	374
11.3.2	Choosing set-up options for a JavaHelp or Oracle Help project	375
11.3.3	Deciding where to locate configuration settings	376
11.3.4	Specifying output options for JavaHelp	376
11.3.5	Establishing a JavaHelp environment	377
11.3.6	Establishing an Oracle Help environment	377
11.3.7	Creating a directory structure for JavaHelp / Oracle Help	378
11.3.8	Configuring the helpset file	382
11.3.9	Coping with JavaHelp / Oracle Help viewer limitations	384
11.3.10	Compiling JavaHelp with Helen	384
11.4	Generating contents and index	385
11.4.1	Configuring contents entries for JavaHelp or Oracle Help	385
11.4.2	Assigning TOC images and expansion levels in JavaHelp 2	385
11.4.3	Configuring index entries for JavaHelp or Oracle Help	386
11.4.4	Eliminating index-marker artifacts from text	386
11.4.5	Locating JavaHelp or Oracle Help contents and index files	387
11.5	Providing full-text search for JavaHelp / Oracle Help	387
11.5.1	Including a search-index link in the helpset file	387
11.5.2	Creating a search index for JavaHelp	388
11.5.3	Creating a search index for Oracle Help	389

11.6	Creating and viewing a Java Archive (JAR) file	390
11.6.1	Creating a JAR file.	391
11.6.2	Viewing a JAR file.	391
11.7	Converting a glossary to JavaHelp 2	392
11.7.1	Evaluating glossary usability	392
11.7.2	Assigning glossary properties	392
11.7.3	Configuring glossary IDs.	392
11.7.4	Eliminating glossary entries from the JavaHelp TOC	393
11.8	Defining windows for JavaHelp or Oracle Help.	393
11.8.1	Specifying window parameters for JavaHelp 2	393
11.8.2	Specifying window parameters for Oracle Help	398
11.8.3	Jumping to secondary windows in JavaHelp or Oracle Help	399
11.9	Linking to destinations within topics	399
11.10	Creating ALinks for Oracle Help	399
11.11	Merging JavaHelp or Oracle Help systems	400
11.12	Setting up CSH for JavaHelp or Oracle Help.	401
12	Generating Eclipse Help	403
12.1	Understanding how Eclipse Help works.	403
12.2	Setting up an Eclipse Help project	403
12.2.1	Creating an Eclipse Help project	403
12.2.2	Choosing set-up options for an Eclipse Help project.	404
12.2.3	Deciding where to locate configuration settings	405
12.2.4	Specifying Eclipse Help output options.	405
12.2.5	Making sure links work in Eclipse Help	406
12.2.6	Disabling breadcrumb trails in Eclipse Help	406
12.3	Configuring Eclipse Help manifest files.	407
12.3.1	Specifying a Java manifest file for Eclipse Help	407
12.3.2	Specifying Eclipse Help plug-in properties	407
12.3.3	Configuring the Java manifest file for Eclipse Help	408
12.3.4	Configuring the plug-in manifest file for Eclipse Help	409
12.4	Configuring contents and index for Eclipse Help.	411
12.4.1	Choosing contents and index methods for Eclipse Help	411
12.4.2	Supplying path information for contents and index links	412
12.4.3	Encoding special characters for contents and index entries.	412
12.4.4	Configuring contents properties for Eclipse Help	412
12.4.5	Configuring index properties for Eclipse Help	414
12.5	Configuring search properties for Eclipse Help	415
12.6	Merging Eclipse Help projects	415
12.6.1	Linking primary content to secondary TOCs.	415
12.6.2	Linking secondary TOCs to primary content (<i>deprecated</i>)	416
12.7	Setting up CSH for Eclipse Help	417
12.7.1	Understanding how Mif2Go generates context links	417
12.7.2	Naming context file and attribute for secondary plug-ins	417
12.7.3	Configuring context IDs and context anchors	418
12.7.4	Configuring context descriptions.	418
12.7.5	Locating context information.	419
12.8	Packaging Eclipse Help files	419
12.8.1	Specifying a ZIP command for doc.zip	419

12.8.2	Specifying ZIP command parameters	419
12.8.3	Specifying a JAR command for doc.jar	420
12.8.4	Monitoring the packaging step for errors	420
12.8.5	Archiving Eclipse Help files	420
13	Converting to HTML/XHTML	423
13.1	Deciding which type of output to produce	424
13.2	Setting up an HTML project	424
13.2.1	Creating an HTML or XHTML project.	425
13.2.2	Choosing set-up options for an HTML or XHTML project	425
13.2.3	Preparing a document for conversion to HTML or XHTML	426
13.2.4	Specifying a different output file extension	427
13.2.5	Checking automatic settings for HTML or XML split files	427
13.2.6	Establishing a conversion workflow for HTML	427
13.2.7	Checking HTML output files for broken links	428
13.2.8	Checking HTML or XML output files for Mif2Go version	428
13.2.9	Using XHTML tagging rules for HTML.	428
13.3	Including starting code and entity references.	429
13.4	Supplying values for the <head> element	429
13.4.1	Specifying HTML/XML version, DOCTYPE, and DTD	429
13.4.2	Specifying namespace and language	430
13.4.3	Specifying character encoding for HTML	431
13.4.4	Including or omitting HTML/XML generator information.	433
13.4.5	Specifying page titles for HTML output files	433
13.4.6	Supplying content for the <meta> tag	434
13.4.7	Specifying nonstandard values for declarations	435
13.5	Specifying HTML <body> attributes.	436
13.6	Specifying document-wide properties for HTML	436
13.6.1	Specifying a default DPI setting	436
13.6.2	Converting system variables to text for HTML	437
13.6.3	Suppressing closing </p> tags for HTML.	437
13.6.4	Suppressing line breaks in HTML and XML output.	437
13.6.5	Preventing adjacent <pre> elements from merging.	438
13.7	Defining and mapping colors for HTML	438
13.7.1	Converting colors	438
13.7.2	Mapping FrameMaker colors to new values	439
13.7.3	Defining new colors	440
13.7.4	Using Web-safe colors	440
13.7.5	Redefining colors via conversion template	440
13.7.6	Understanding CMYK-to-RGB conversion anomalies.	441
13.8	Converting generated files for HTML	441
13.8.1	Converting FrameMaker IX and other marker lists	442
13.8.2	Converting FrameMaker TOC and other paragraph lists	444
13.9	Importing HTML files as insets	446
13.10	Converting conditions to HTML attributes	446
13.10.1	Understanding how Mif2Go converts conditions	446
13.10.2	Mapping FrameMaker conditions to HTML attributes	447
13.10.3	Displaying condition indicators in HTML with CSS	447
13.11	Providing hover text for terms in HTML	448
13.12	Generating XHTML for Confluence 4.x	449

13.13	Exporting content for database input	450
13.14	Using framesets	450
13.15	Adding a “Made with Mif2Go” label or button	452
13.16	Passing W3C validation tests	453
13.16.1	Understanding limitations of W3C validation	453
13.16.2	Replacing high ASCII characters for W3C validation	454
13.16.3	Eliminating <nobr> tags	455
13.16.4	Removing full-row straddles from tables	456
13.16.5	Avoiding redundant attribute assignments in tables	456
13.16.6	Eliminating duplicate ObjectIDs	456
14	Converting to generic XML	457
14.1	Understanding how Mif2Go generates XML output	457
14.1.1	Accommodating HTML features in XML output	457
14.1.2	Introducing structure with Mif2Go	458
14.1.3	Introducing structure with XSLT	458
14.1.4	Creating structure in FrameMaker	458
14.1.5	Producing SGML with Mif2Go and XSLT	458
14.2	Setting up a generic XML project	459
14.3	Specifying generic XML output settings	459
14.3.1	Creating a generic XML project	460
14.3.2	Changing output XML version or file extension	460
14.3.3	Specifying character encoding for generic XML	460
14.3.4	Specifying the root element and content type	461
14.3.5	Preventing arbitrary line breaks in XML text elements	461
14.4	Providing XML tags and structure	461
14.4.1	Generating XML from an unstructured document	462
14.4.2	Deriving XML tags from format and class names	462
14.4.3	Eliminating HTML attributes and tags for generic XML	463
14.4.4	Including or excluding FrameMaker autonumbers	465
14.4.5	Configuring forced returns for XML	465
14.5	Converting FrameMaker lists to generic XML	466
14.6	Configuring links for generic XML	467
14.7	Converting graphics for generic XML	468
14.8	Converting index entries to generic XML	468
14.8.1	Configuring index markers for conversion to XML	469
14.8.2	Defining macros to process index content	469
15	Converting to DITA XML	473
15.1	Generating DITA XML with Mif2Go	473
15.1.1	Understanding the complexity of a DITA conversion project	473
15.1.2	Understanding what you need to know about DITA	474
15.1.3	Clarifying your purpose for creating DITA output	474
15.1.4	Converting from structured vs. unstructured FrameMaker	475
15.1.5	Understanding what information you must supply	476
15.1.6	Understanding how Mif2Go generates DITA output	476
15.1.7	Creating valid DITA XML output	477
15.2	Setting up a DITA XML project	478
15.2.1	Creating a DITA XML project	478
15.2.2	Choosing set-up options for a DITA XML project	479

15.2.3	Specifying DITA output options	480
15.2.4	Specifying DITA version	480
15.2.5	Configuring the DITA DTD SYSTEM identifier	481
15.2.6	Ensuring FrameMaker 8 import compatibility	481
15.2.7	Substituting document format names for default names	481
15.3	Specifying general options for DITA	483
15.4	Configuring DITA elements	486
15.4.1	Understanding how Mif2Go delimits DITA elements	486
15.4.2	Treating FrameMaker format names as DITA element names	486
15.4.3	Mapping paragraph formats to DITA block elements	487
15.4.4	Mapping character formats to DITA inline elements	492
15.4.5	Assigning multiple typographic elements to a format	494
15.4.6	Assigning attributes to DITA elements	495
15.4.7	Preserving whitespace in block elements	499
15.4.8	Including PIs for line, column, or page breaks	499
15.4.9	Providing a <shortdesc> element for a DITA topic	500
15.4.10	Converting index markers to <indexterm> elements	500
15.5	Nesting DITA block elements	501
15.5.1	Understanding how Mif2Go determines element nesting	501
15.5.2	Designating DITA ancestor elements	502
15.5.3	Fixing up interpolated ancestries	503
15.5.4	Deciding when to fully specify ancestry	503
15.5.5	Specifying alternate ancestries for the same element	504
15.5.6	Avoiding invalid ancestries	504
15.5.7	Specifying first-child status for nested elements	505
15.5.8	Configuring nested lists	505
15.5.9	Closing DITA ancestor elements	506
15.5.10	Opening DITA ancestor elements	507
15.5.11	Configuring multi-paragraph list items	508
15.5.12	Splitting a paragraph into separate DITA elements	508
15.5.13	Specifying DITA element levels	509
15.6	Converting tables to DITA XML	510
15.6.1	Working with Mif2Go DITA table types	510
15.6.2	Marking table footer rows for future reference	511
15.6.3	Designating ancestors for <table> elements	512
15.6.4	Applying attributes to DITA tables	512
15.6.5	Configuring DITA table components	513
15.6.6	Converting tables used only as image containers	514
15.6.7	Omitting table coding entirely	515
15.7	Specifying options for images in DITA XML	516
15.7.1	Designating ancestors for <image> and <fig> elements	516
15.7.2	Specifying what to include in a <fig> wrapper	517
15.7.3	Omitting size attributes from images for DITA output	518
15.7.4	Providing alternate text for images	518
15.7.5	Including MathFullForm equations in <alt> elements	518
15.7.6	Including the original image DPI as an attribute	518
15.7.7	Understanding why images might look incorrectly scaled	519
15.8	Organizing DITA topics	519
15.8.1	Understanding when to split, nest, or wrap DITA topics	519
15.8.2	Splitting FrameMaker files into DITA topic files	520
15.8.3	Renaming DITA topic files	520

15.8.4	Nesting DITA topics in unsplit files	521
15.8.5	Wrapping DITA topics in a top-level <dita> element	521
15.9	Configuring DITA topics	522
15.9.1	Designating starting points for DITA topics	522
15.9.2	Specifying the DITA topic type	524
15.9.3	Specifying the ID for a DITA topic	526
15.9.4	Adjusting DITA topic IDs generated from file names	526
15.9.5	Specifying alternate titles for a DITA topic	526
15.9.6	Omitting a DITA topic from the TOC	527
15.10	Configuring cross references and links for DITA	527
15.10.1	Understanding how Mif2Go converts cross references	527
15.10.2	Specifying an outputclass for cross-reference wrappers	528
15.10.3	Linking to elements below topic level	528
15.10.4	Retaining cross-reference content in <xref> elements	528
15.10.5	Omitting <xref> elements from footnotes	529
15.10.6	Overriding <xref> attribute values	529
15.11	Exporting FrameMaker variables to DITA XML	530
15.11.1	Understanding how Mif2Go represents variables in DITA	530
15.11.2	Specifying how to treat FrameMaker variables	530
15.11.3	Treating FrameMaker variables as conrefs	531
15.11.4	Retaining format properties of user variables in DITA	532
15.12	Converting conditions to DITA attributes	533
15.12.1	Understanding how Mif2Go converts conditional text	533
15.12.2	Mapping FrameMaker conditions to element attributes	533
15.12.3	Disallowing condition conversion for selected elements	534
15.13	Marking FrameMaker text insets in DITA	534
15.14	Including CSH targets in DITA XML	535
15.15	Overriding DITA settings with markers	536
16	Configuring DITA maps	539
16.1	Understanding how Mif2Go generates DITA maps	539
16.2	Configuring DITA ditamaps	539
16.2.1	Specifying options for ditamaps	539
16.2.2	Specifying topic levels in ditamaps	544
16.2.3	Accounting for missing topic levels	544
16.2.4	Specifying roles for topics in ditamaps	545
16.2.5	Adding relationship tables to ditamaps	546
16.2.6	Providing navigation aids in ditamaps	547
16.3	Constructing a DITA bookmap	548
16.3.1	Specifying the type of map for a book	548
16.3.2	Specifying <booktitle> information	548
16.3.3	Specifying <bookmeta> information	549
16.3.4	Extending <part> to include <appendix>	550
16.3.5	Choosing whether a bookmap references maps or topics	550
16.3.6	Excluding the book-level reltable from a bookmap	550
16.4	Mapping FrameMaker files to bookmap components	551
16.4.1	Assigning bookmap roles to FrameMaker files	551
16.4.2	Assigning frontmatter and backmatter roles and components	552
16.4.3	Including multiple booklist components of the same type	553
16.4.4	Assigning a divider role to a section file or chapter	554

16.4.5	Assigning a series of roles to a single FrameMaker file	554
16.4.6	Assigning a single role to a series of FrameMaker files	554
16.4.7	Including placeholders for additional bookmap elements.	555
16.5	Providing attributes for bookmap wrapper elements	555
16.6	Overriding DITA map settings with markers.	556
17	Converting to DocBook XML	557
17.1	Generating DocBook XML with Mif2Go	557
17.1.1	Understanding what you need to know about DocBook	557
17.1.2	Clarifying your purpose for creating DocBook output	557
17.1.3	Understanding what information you must supply	558
17.2	Setting up a DocBook XML project.	559
17.2.1	Creating a DocBook project	559
17.2.2	Choosing set-up options for a DocBook project	560
17.2.3	Specifying DocBook output options	561
17.3	Specifying general options for DocBook.	562
17.3.1	Configuring styles for DocBook XML	562
17.3.2	Configuring entity information for DocBook XML	563
17.3.3	Configuring links for DocBook XML.	563
17.3.4	Configuring tables for DocBook XML	563
17.3.5	Retaining empty paragraph tags in DocBook table cells.	564
17.3.6	Configuring footnotes for DocBook XML	564
17.4	Configuring DocBook elements.	564
17.4.1	Treating FrameMaker format names as element names	565
17.4.2	Mapping paragraph formats to DocBook elements.	565
17.4.3	Mapping character formats to DocBook elements	568
17.4.4	Assigning ID attributes to DocBook block elements	569
17.4.5	Assigning attributes other than ID to DocBook elements.	571
17.5	Nesting DocBook block elements	573
17.5.1	Understanding how Mif2Go determines element nesting.	573
17.5.2	Designating DocBook ancestor elements	573
17.5.3	Fixing up interpolated ancestries.	574
17.5.4	Deciding when to fully specify ancestry	575
17.5.5	Specifying alternate ancestries for the same element	575
17.5.6	Specifying first-child status for nested elements.	576
17.5.7	Specifying full ancestry for nested sections	576
17.5.8	Closing DocBook ancestor elements.	577
17.5.9	Opening DocBook ancestor elements	578
17.5.10	Configuring multi-paragraph list items	578
17.5.11	Specifying DocBook element levels	579
17.6	Designating ancestors for table elements.	580
17.7	Specifying options for figure elements	581
17.7.1	Deciding what to include in a figure element	581
17.7.2	Specifying ancestry for figure elements	581
17.7.3	Omitting size attributes from images for DocBook	582
17.8	Overriding DocBook settings with markers.	582
18	Splitting and extracting files	585
18.1	Splitting versus extracting	585
18.2	Splitting files	586

18.2.1	Designating split points	586
18.2.2	Managing split points.	588
18.2.3	Combining instead of splitting files.	591
18.3	Extracting files	591
18.3.1	Enabling and disabling extract processing.	591
18.3.2	Delimiting material to extract	592
18.4	Identifying split and extract files	593
18.4.1	Understanding how split and extract files are named	593
18.4.2	Specifying page titles for split or extract files	594
18.4.3	Supplying <meta> text for split or extract files	598
18.5	Inserting HTML code in split and extract files.	598
18.5.1	Choosing how to insert code in extracts	598
18.5.2	Assigning code to [Inserts] keywords for splits and extracts.	599
18.5.3	Using special sections to insert code in extracts	600
18.6	Referencing split and extract files.	600
18.7	Customizing and replacing extracts	601
18.7.1	Using markers for extract processing.	602
18.7.2	Customizing title text for extracts	602
18.7.3	Replacing extracts with links in the parent file	603
18.7.4	Specifying extracts: an example	607
19	Creating HTML links	609
19.1	Understanding sources of links.	609
19.2	Specifying link appearance.	609
19.2.1	Specifying link colors	610
19.2.2	Specifying link class	610
19.2.3	Assigning link attributes with markers	612
19.2.4	Specifying link properties with macros	612
19.2.5	Replacing problem characters in links.	612
19.2.6	Forcing link text to lowercase	613
19.3	Specifying link destination	613
19.3.1	Forcing links to top-of-page for selected paragraph formats.	614
19.3.2	Forcing all links to top-of-page	614
19.3.3	Linking to an arbitrary location	614
19.3.4	Providing alternate link destinations	615
19.3.5	Troubleshooting bad links	616
19.4	Creating jumps to particular windows for HTML	616
19.5	Converting FrameMaker links to HTML	617
19.5.1	Converting FrameMaker cross references to HTML.	617
19.5.2	Converting FrameMaker hypertext links to HTML.	619
19.5.3	Including ObjectID anchors as link targets	620
19.6	Linking to other files and other Mif2Go projects.	621
19.6.1	Identifying HTML link destinations with FileIDs.	621
19.6.2	Retaining file paths in interfile links	622
19.6.3	Enabling links to renamed or relocated files	622
19.6.4	Enabling links to files in other projects	623
19.6.5	Updating links between files in different projects	624
19.6.6	Mapping links to text insets	624
19.7	Linking to external destinations	625

20	Providing navigation in HTML	627
20.1	Understanding how navigation links work	627
20.2	Generating trails of links	627
20.2.1	Understanding trails of links	627
20.2.2	Specifying whether to include trails of links	628
20.2.3	Specifying what to include in trails of links	628
20.2.4	Specifying heading levels for trails of links	630
20.2.5	Specifying where to display trails of links	630
20.3	Including local TOCs	631
20.3.1	Directing Mif2Go to generate local TOCs	631
20.3.2	Configuring local TOCs	631
20.3.3	Positioning local TOCs in HTML topics	634
20.3.4	Creating local TOCs in FrameMaker	635
20.4	Creating a browse sequence	635
20.4.1	Understanding how browse macros work	636
20.4.2	Choosing buttons versus text links for a browse sequence	638
20.4.3	Formatting browse-link labels	639
20.4.4	Modifying macros <\$_prev>, <\$_next>, and <\$_top>	639
20.4.5	Understanding browse keyword scope and default values	641
20.4.6	Specifying where to invoke a browse macro	642
20.4.7	Considering an example of browse navigation	643
20.4.8	Specifying an alternate file sequence for browse links	644
21	Mapping text formats to HTML/XML	645
21.1	Understanding how Mif2Go converts text	645
21.2	Choosing how to map formats	645
21.3	Mapping paragraph formats	646
21.3.1	Assigning HTML tags and attributes to paragraph formats	646
21.3.2	Converting sidehead and run-in paragraph formats	648
21.3.3	Converting paragraph formats with autonumbers	648
21.3.4	Including text-frame content in line	649
21.3.5	Designating script paragraph formats	650
21.3.6	Stripping paragraph properties	650
21.3.7	Keeping or removing reference frames	651
21.3.8	Deciding how to treat forced returns	651
21.3.9	Providing content for empty paragraphs	651
21.3.10	Eliminating empty paragraphs in text	652
21.3.11	Eliminating invisible paragraphs	652
21.3.12	Eliminating unwanted paragraphs	652
21.4	Mapping character formats	653
21.5	Assigning properties to text formats	653
21.5.1	Understanding where to specify format property overrides	654
21.5.2	Overriding paragraph alignment and size properties	656
21.5.3	Overriding properties added by typographic elements	657
21.5.4	Overriding properties specified in font tags	657
21.6	Mapping special characters	658
21.6.1	Understanding how Mif2Go represents characters	658
21.6.2	Understanding how Mif2Go treats tabs in HTML/XML	658
21.6.3	Understanding Mif2Go support for FrameMaker 8+ Unicode	659
21.6.4	Converting Western European accented characters	660

21.6.5 Mapping individual special characters.	660
21.6.6 Mapping characters in a special font	662
21.6.7 Avoiding use of special characters in URIs.	663
21.6.8 Preventing character mapping	663
21.7 Mapping fonts	663
21.7.1 Specifying a default font and size	664
21.7.2 Remapping fonts	664
21.7.3 Mapping font sizes.	664
21.7.4 Including or excluding font tags	665
21.7.5 Managing font tags for symbol fonts.	666
21.7.6 Excluding face and size attributes from font tags	666
21.7.7 Accommodating browser font-rendering differences	666
21.8 Managing typographic elements for HTML or XML.	667
21.8.1 Deciding whether to suppress typographic elements.	667
21.8.2 Choosing how to treat typographic elements.	667
21.9 Specifying text colors for HTML	669
21.10 Configuring preformatted text for HTML/XML	670
21.10.1 Eliminating line wraps in preformatted text	670
21.10.2 Replacing tabs with spaces in preformatted text	671
21.11 Converting footnotes to HTML or XML	671
21.11.1 Configuring and placing footnotes.	671
21.11.2 Eliminating links to jump footnotes.	672
21.11.3 Using list tags or <div> and <p> tags for jump footnotes	672
21.11.4 Formatting jump footnote text with macros.	673
21.12 Converting list formats to HTML.	674
21.12.1 Understanding the problem with HTML lists	674
21.12.2 Converting list formats to HTML list styles	675
21.12.3 Indenting list items.	678
21.12.4 Converting list formats to HTML/XML paragraphs	679
22 Setting up CSS for HTML	681
22.1 Deciding whether to use CSS	681
22.2 Understanding how to use CSS.	681
22.3 Understanding how Mif2Go generates CSS.	682
22.4 Specifying CSS file and link options	683
22.4.1 Specifying CSS options at project set-up time.	683
22.4.2 Specifying CSS options in a Mif2Go configuration file	684
22.4.3 Designating and locating a CSS file	686
22.4.4 Directing Mif2Go to generate a CSS file	686
22.4.5 Understanding effects of the older Stylesheet setting	687
22.5 Understanding how CSS affects other options	687
22.6 Linking to alternate CSS files.	688
22.6.1 Selecting a CSS file at run time	688
22.6.2 Changing CSS files in the middle of a document	689
22.6.3 Customizing the CSS link tag	690
22.6.4 Using an alternate CSS link tag for Netscape 4.	690
22.7 Assigning CSS classes	691
22.7.1 Understanding CSS class name restrictions.	691
22.7.2 Mapping paragraph formats to CSS classes.	692
22.7.3 Mapping character formats to tags or span classes	693

22.7.4	Assigning CSS classes to table formats.	694
22.7.5	Assigning CSS classes to text and table footnotes	694
22.7.6	Assigning CSS classes based on Unicode character ranges	694
22.7.7	Assigning CSS classes to FrameMaker conditions	695
22.7.8	Using link format names as CSS class names.	696
22.7.9	Using CSS class names as tags for XML	696
22.7.10	Omitting tags from CSS selectors	696
22.7.11	Overriding CSS class for selected paragraphs.	697
22.8	Customizing CSS properties	698
22.8.1	Assigning a CSS generic font family	698
22.8.2	Specifying CSS <body> tag properties	698
22.8.3	Specifying CSS size values and units of measurement.	699
22.8.4	Overriding styles in Mif2Go -generated CSS files	700
22.8.5	Adjusting leading (line spacing) in CSS	700
22.8.6	Preventing tags from overriding CSS properties.	701
23	Including graphics in HTML	703
23.1	Starting with default graphics options	703
23.2	Understanding graphics processing for HTML	703
23.3	Locating graphics files for HTML	704
23.4	Specifying options for HTML graphics	705
23.4.1	Using referenced graphics without converting	706
23.4.2	Specifying formats of replacement graphics.	706
23.4.3	Choosing a graphics conversion method.	707
23.4.4	Using referenced, embedded, and compound graphics.	707
23.4.5	Omitting graphics from HTML or XML output	708
23.5	Selecting and modifying graphics	708
23.5.1	Assigning properties to sets of graphics	708
23.5.2	Replacing or surrounding a graphic with macro code.	710
23.5.3	Converting only the visible portion of a graphic.	712
23.5.4	Converting reference-page graphics for HTML	712
23.5.5	Eliminating graphics in unanchored frames	713
23.5.6	Omitting paragraph tags around graphics	713
23.5.7	Retaining run-in images in otherwise empty paragraphs	713
23.6	Positioning graphics in HTML output	714
23.6.1	Positioning graphics anchored in empty paragraphs.	714
23.6.2	Aligning anchored graphics.	714
23.6.3	Indenting images	716
23.6.4	Adding space above an image.	717
23.6.5	Eliminating space above or below graphics in table cells.	717
23.7	Specifying HTML image attributes	718
23.8	Providing (or omitting) alternate text for images.	718
23.9	Scaling images for HTML	719
23.9.1	Excluding image size attributes from HTML	720
23.9.2	Adjusting image size for selected graphics	720
23.9.3	Adjusting image resolution for referenced graphics	721
23.9.4	Specifying image resolution for exported graphics.	721
23.9.5	Specifying px units for graphics sized in pixels	722
23.10	Creating image maps for HTML	722
23.10.1	Creating hotspots for image maps	722

23.10.2	Providing alternate text for a hotspot in an image map	723
23.10.3	Specifying jumps from image maps in framesets	725
23.11	Supplying a background image or watermark	725
23.12	Converting equations for HTML	725
24	Converting tables to HTML	727
24.1	Assigning properties to tables	727
24.1.1	Understanding which table features can be converted	727
24.1.2	Understanding precedence of assignment methods	728
24.1.3	Overriding default table and cell properties and attributes	728
24.2	Defining sets of tables	728
24.2.1	Determining the TableID	729
24.2.2	Creating table groups	729
24.2.3	Using wildcards to specify table sets	730
24.3	Specifying table structure	730
24.3.1	Choosing the table structure model	730
24.3.2	Identifying row and column groups and header cells	731
24.3.3	Identifying table headers and footers	734
24.4	Specifying table attributes	735
24.4.1	Specifying attributes for all tables	736
24.4.2	Overriding attributes for selected tables	736
24.4.3	Assigning a CSS class to a table	737
24.4.4	Using markers to assign attributes to tables, rows, or cells	737
24.4.5	Specifying attributes for table rows	737
24.4.6	Specifying attributes for table cells	738
24.4.7	Eliminating automatically generated attributes	739
24.4.8	Adjusting borders, cell spacing, and cell padding	739
24.4.9	Determining the width of table columns	741
24.4.10	Deciding what to do with empty paragraphs in table cells	744
24.4.11	Using shading and color in tables	745
24.5	Positioning tables, table titles, and table footnotes	746
24.5.1	Indenting tables	747
24.5.2	Configuring and positioning table titles	747
24.5.3	Eliminating FrameMaker table title variables	748
24.5.4	Positioning table footnotes	748
24.6	Using macros to control table properties	748
24.6.1	Invoking macros around tables	748
24.6.2	Adding space before tables	749
24.6.3	Adjusting space after tables	749
24.6.4	Turning processing on and off around selected tables	750
24.6.5	Specifying row-group, row, and cell attributes with macros	750
24.6.6	Capturing table row and column counts with variables	751
24.6.7	Selectively modifying table text with macros: an example	752
24.7	Converting tables to paragraphs	753
24.7.1	Removing table-specific tags from all tables	753
24.7.2	Removing table-specific tags from selected tables	754
24.7.3	Removing table-specific tags from complex tables	754
25	Generating WAI markup for HTML	755
25.1	Comparing Mif2Go markup methods for WAI	755
25.1.1	Choosing a markup method for WAI attributes	755

25.1.2	Using paragraph formats for WAI attributes	755
25.1.3	Creating custom markers for WAI attributes	756
25.2	Applying WAI markup to images	756
25.2.1	Following WAI guidelines for images	757
25.2.2	Assigning WAI image attributes with dedicated formats	757
25.2.3	Assigning WAI image attributes with custom markers.	757
25.2.4	Assigning WAI image attributes via the <i>Object Attributes</i> dialog	758
25.3	Applying WAI markup to links	758
25.3.1	Following WAI guidelines for links	758
25.3.2	Assigning WAI link attribute values with dedicated formats	758
25.3.3	Assigning WAI link attribute values with custom markers.	759
25.4	Applying WAI markup to tables	759
25.4.1	Following WAI guidelines for tables	759
25.4.2	Choosing a WAI markup method for tables	760
25.4.3	Providing table summary and title information.	760
25.4.4	Identifying table row and column information	762
26	Identifying HTML table structure for WAI	763
26.1	Identifying table rows and columns	763
26.1.1	Developing a strategy for row and column markup	763
26.1.2	Comparing scope and id/headers accessibility methods	763
26.1.3	Specifying a default accessibility method.	764
26.1.4	Overriding the default accessibility method	765
26.2	Associating table cells with header cells	766
26.2.1	Specifying group properties for header cells.	766
26.2.2	Using paragraph formats for table-cell attributes	767
26.2.3	Assigning table-cell attribute values with dedicated formats	772
26.2.4	Assigning table-cell attribute values with custom markers.	772
27	Marking HTML table cells for WAI	775
27.1	Understanding table cell settings	775
27.2	Using the scope method to identify table cells.	775
27.3	Using the id/headers method to identify table cells	777
27.3.1	Choosing an id/headers level.	777
27.3.2	Specifying id/headers attributes for table cells	777
27.3.3	Grouping header cells for identification	778
27.3.4	Column-group and row-group extent	779
27.3.5	Choosing a different row-group method	780
27.3.6	Using span attributes to identify rows and columns	780
27.3.7	Column-span and row-span extent	781
27.3.8	Identifying individual table cells by row and column.	782
27.3.9	Column and row extent	783
27.3.10	Using span IDs with row or column IDs.	783
27.4	Overriding default table-cell settings	784
27.5	Using ColGroup and RowGroup cells	784
27.5.1	Understanding how the ColGroup property works	784
27.5.2	Understanding how the RowGroup property works	785
28	Working with macros	787
28.1	Defining and invoking macros	787
28.1.1	Defining macros	787

28.1.2	Invoking a macro	791
28.1.3	Nesting macros.	791
28.1.4	Using predefined macros	792
28.2	Accessing Mif2Go macro libraries.	792
28.2.1	Understanding Mif2Go -supplied macro libraries	792
28.2.2	Modifying Mif2Go -supplied macro definitions	793
28.2.3	Storing a macro definition in a separate file	793
28.2.4	Including macro definitions in your own macro library	794
28.3	Using macro variables.	795
28.3.1	Creating and invoking macro variables	796
28.3.2	Assigning values to macro variables	797
28.3.3	Incrementing and decrementing macro variables	799
28.3.4	Using predefined macro variables	800
28.3.5	Treating FrameMaker user variables as macro variables.	801
28.3.6	Using some FrameMaker system variables as macro variables.	802
28.3.7	Creating macro variables from paragraph content.	802
28.4	Using multiple-value list variables	806
28.4.1	Understanding list-variable syntax.	806
28.4.2	Assigning a value to a list-variable item	806
28.4.3	Initializing list variables.	807
28.4.4	Using macros to process lists.	807
28.4.5	Using pointers to process lists	808
28.4.6	Using a list instead of a conditional expression.	809
28.5	Accessing settings with configuration macros	809
28.5.1	Understanding configuration macros and variables.	809
28.5.2	Determining the value of a configuration variable	810
28.5.3	Deploying configuration macros	810
28.6	Using expressions in macros.	811
28.6.1	Understanding macro expressions	811
28.6.2	Understanding operands and operators	811
28.6.3	Displaying expression results in output	813
28.6.4	Using control structures in expressions	815
28.6.5	Specifying substrings in expressions	817
28.6.6	Using list variables in expressions.	818
28.6.7	Using indirection in expressions	819
28.6.8	Removing spaces from strings: an example.	820
28.7	Passing a parameter to a macro.	820
28.8	Debugging macros	820
28.9	Deploying macros and macro variables	820
28.9.1	Understanding where to use macros and macro variables.	821
28.9.2	Invoking macros at predetermined points in output.	821
28.9.3	Surrounding or replacing text with code or macros.	822
28.9.4	Converting a dictionary-style list to an HTML table.	824
28.9.5	Assigning macros to graphics or tables for HTML	827
28.9.6	Redefining navigation macros in HTML.	827
28.9.7	Using HTML Macro markers to invoke macros	828
28.9.8	Implementing drop-down text with macros.	828
28.10	Using macros to fine-tune HTML or XML output.	828

29 Working with FrameMaker markers	831
29.1 Using custom FrameMaker markers	831
29.2 Adding custom marker types	832
29.2.1 Identifying dedicated custom marker types	832
29.2.2 Naming new custom marker types	834
29.2.3 Understanding attribute markers	834
29.2.4 Using attribute markers for HTML or XML	835
29.3 Remapping marker types and hypertext commands	836
29.3.1 Remapping and cloning marker types	836
29.3.2 Understanding when to remap marker types	837
29.3.3 Remapping FrameMaker hypertext commands	837
29.4 Defining and redefining marker behavior	838
29.4.1 Assigning properties to marker types	838
29.4.2 Observing restrictions on redefining marker behavior	840
29.4.3 Understanding examples of marker redefinition	840
29.5 Suppressing markers	841
29.6 Using marker property names for marker types	842
29.7 Inserting code or text with markers	842
29.7.1 Inserting marker content in output	842
29.7.2 Surrounding marker content with code	843
29.7.3 Processing marker content as text for XML/HTML/XHTML	844
29.7.4 Surrounding attribute markers with code	845
29.7.5 Converting custom markers to attributes	845
29.7.6 Including code to be executed before a topic	846
29.8 Identifying markers with variable <\$\$_objectid>	847
30 Working with templates	849
30.1 Working with configuration templates	849
30.1.1 Understanding how templates are organized	849
30.1.2 Understanding how templates are named	850
30.1.3 Understanding how templates are chained together	850
30.1.4 Understanding how macro libraries are organized	851
30.2 Referencing configuration files and templates	851
30.3 Including document-specific configuration files	852
30.3.1 Understanding document-specific configuration files	853
30.3.2 Referencing a document-specific configuration file	853
30.3.3 Deciding where to keep document-specific configuration files	854
30.3.4 Indicating the intended scope of a configuration file	855
30.4 Including chapter-specific configuration files	855
30.5 Deciding which configuration file to edit	856
30.5.1 Understanding what configuration files are available	857
30.5.2 Editing a project configuration file	858
30.5.3 Editing a document-specific configuration file	859
30.5.4 Editing an output-specific configuration file	860
30.5.5 Editing a macro configuration file	861
30.5.6 Indicating the intended scope of a configuration file	861
30.6 Creating your own configuration templates	861
30.6.1 Creating a template from a project configuration file	862
30.6.2 Deciding what to include in a general configuration template	862

30.6.3	Chaining configuration templates	863
30.7	Applying FrameMaker conversion templates.	863
30.7.1	Specifying conversion-template settings	864
30.7.2	Applying alternate conversion templates.	865
30.7.3	Changing template options.	866
30.7.4	Avoiding template-related disasters.	866
30.7.5	Troubleshooting template import problems.	866
31	Working with graphics	869
31.1	Choosing an appropriate graphics format.	869
31.1.1	Graphics formats for Word documents	869
31.1.2	Graphics formats for WinHelp.	869
31.1.3	WMF format limitations	870
31.1.4	Graphics formats for HTML	871
31.2	Converting and exporting graphics.	871
31.2.1	Converting bitmap graphics.	871
31.2.2	Converting vector graphics	874
31.2.3	Exporting and converting embedded graphics.	877
31.2.4	Exporting images and creating files from OLE objects.	881
31.2.5	Converting graphics with FrameMaker export filters	883
31.2.6	Embedding bitmap graphics in WMF for WinHelp.	886
31.2.7	Exporting embedded graphics imported from Word	886
31.3	Replacing and relocating graphics files	887
31.3.1	Changing graphics files for HTML output	887
31.3.2	Changing graphics files for RTF output	890
31.4	Specifying custom settings for individual graphics	895
31.4.1	Overriding graphics settings with custom markers	895
31.4.2	Overriding graphics settings with FrameMaker object attributes	896
31.5	Controlling image appearance in RTF output.	898
31.5.1	Rescaling bitmap graphics	898
31.5.2	Reorienting bitmap graphics	899
31.5.3	Compressing bitmap graphics	899
31.5.4	Positioning borders around inline graphics	900
31.5.5	Mapping FrameMaker pen style patterns.	900
31.5.6	Converting graphic text	901
31.5.7	Specifying transparency for WinHelp 4.	903
31.6	Converting graphics with Microsoft Word filters.	904
32	Working with content models	905
32.1	Understanding Mif2Go content models	905
32.2	Modifying or replacing a content model.	905
32.2.1	Obtaining a copy of a built-in content-model	906
32.2.2	Generating a content model from a DTD.	906
32.3	Preparing a content model for use with Mif2Go	907
32.4	Understanding content-model configurations.	908
32.4.1	Content model [Topic] settings	909
32.4.2	Content model [ElementSets] settings.	910
32.4.3	Content model [TopicParents] settings	910
32.4.4	Content model [TopicFirst] settings	910
32.4.5	Content model [TopicLevels] settings.	911

32.5	Understanding how Mif2Go uses content models	911
32.6	Inspecting and correcting element types	912
32.7	Specializing or modifying DITA topic types	913
32.7.1	Creating a content model for a specialized topic type	913
32.7.2	Overriding settings in a DITA content model	914
32.7.3	Eliminating elements from a DITA content model	915
32.7.4	Overriding declarations in a DITA map content model	915
32.7.5	Listing DITA topic type configuration files	915
32.7.6	Locating DITA topic type configuration files	916
32.7.7	Providing table structure information for DITA topic types	916
32.8	Extracting content-model debug information	918
33	Overriding configuration settings	919
33.1	Using a different configuration for selected files	919
33.1.1	Providing configuration files for individual chapters	919
33.1.2	Understanding precedence of configuration settings	919
33.1.3	Updating a single chapter of a FrameMaker book	920
33.2	Overriding settings with markers or macros	920
33.2.1	Determining the extent of a configuration override	921
33.2.2	Overriding settings with configuration markers	921
33.2.3	Overriding settings with macros	921
33.2.4	Assigning values to configuration variables	922
33.2.5	Adding a new configuration setting on the fly	923
33.2.6	Assigning a macro or variable to a configuration variable	923
33.2.7	Understanding fixed-key vs. variable-key settings	923
33.2.8	Overriding fixed-key configuration settings	924
33.2.9	Overriding variable-key configuration settings	925
33.2.10	Assigning HTML table and graphic groups with overrides	930
33.3	Overriding configuration settings with text	931
34	Automating Mif2Go conversions	933
34.1	Preparing documents for single-sourcing	933
34.1.1	Using character formats to identify Help elements	933
34.1.2	Using markers to add links and instructions	935
34.1.3	Using conditional text to differentiate output	936
34.1.4	Importing formats and conditional text settings	936
34.2	Converting a single chapter of a book	937
34.3	Considering ways to automate conversions	937
34.4	Executing operating-system commands	937
34.4.1	Specifying system commands	938
34.4.2	Including macros and variables in system commands	939
34.4.3	Monitoring system command execution	939
34.4.4	Changing configuration settings with system commands	940
34.4.5	Supplying system commands in a .bat file	940
34.4.6	Supplying system commands in a macro	940
34.5	Supplying run-time values for user variables	941
34.5.1	Assigning an initial value to a user variable	941
34.5.2	Assigning a prompt to a user variable	942
34.5.3	Deciding how often to prompt for values of user variables	942
34.5.4	Understanding when Mif2Go prompts for user variables	942

34.5.5	Inspecting and editing values of user variables	943
34.6	Supporting document review in Word	943
34.7	Converting autonumbers for database systems.	944
34.8	Renaming output files for automated systems	946
34.8.1	Understanding which files can be renamed	946
34.8.2	Renaming individual output files.	946
34.8.3	Using custom markers to name output files.	947
34.8.4	Using paragraph formats to name output files.	947
34.8.5	Including identifiers and sequence numbers in file names	952
35	Producing deliverable results	955
35.1	Understanding Mif2Go pre- and post-processing	955
35.2	Activating and logging production of deliverables.	956
35.3	Understanding path values for deliverables	957
35.4	Clearing out old files before converting	957
35.4.1	Specifying when to delete old files from the project directory	958
35.4.2	Specifying which files to delete from the project directory.	958
35.4.3	Understanding when not to delete .ref and .htm files	959
35.4.4	Deleting MIF files from the project directory	960
35.5	Gathering additional files before converting	960
35.6	Assembling files for distribution	961
35.6.1	Specifying a wrap directory.	961
35.6.2	Emptying the wrap directory before copying	962
35.6.3	Listing files to copy to the wrap directory.	962
35.6.4	Understanding when to use other file copy settings	963
35.6.5	Understanding which files are copied from where	963
35.6.6	Listing extracurricular files to put in the wrap directory	964
35.7	Placing graphics files for distribution.	965
35.7.1	Copying referenced graphics to a distribution directory	965
35.7.2	Selecting graphics to copy from arbitrary locations	966
35.7.3	Deleting prior contents of the graphics destination directory	967
35.7.4	Synchronizing graphics settings for HTML output	968
35.7.5	Synchronizing graphics settings for RTF output	969
35.8	Placing CSS or XSL files for assembly	969
35.9	Gathering files for an HTML project: an example	970
35.10	Gathering and processing Help-system files.	971
35.11	Archiving deliverables	973
35.11.1	Specifying an archiving command.	973
35.11.2	Supplying parameters for the archiving command	973
35.11.3	Specifying archive file name and optional version	974
35.12	Placing deliverables in a shipping directory	975
35.12.1	Specifying a shipping directory for deliverables	975
35.12.2	Understanding which files are placed in the shipping directory	976
35.12.3	Choosing whether to copy or move deliverables.	976
35.13	Postprocessing separately from converting	976
36	Converting via runfm	979
36.1	Designing a project for unattended operation.	979
36.2	Setting up FrameMaker for unattended operation	980

36.3	Understanding runfm command-line syntax	980
36.4	Using runfm for Mif2Go conversions	982
36.4.1	Locating FrameMaker executable and files	982
36.4.2	Identifying your Mif2Go project	983
36.4.3	Configuring runfm output	984
36.4.4	Closing FrameMaker files after conversion	987
36.5	Troubleshooting runfm processes	987
36.5.1	Increasing console diagnostics: runfm -diag option	988
36.5.2	Capturing console diagnostics: runfm -log option	988
36.5.3	Reviewing FrameMaker console messages after runfm	988
36.5.4	Troubleshooting failed runfm processes	989
36.5.5	Running a single Mif2Go conversion or print job	989
36.5.6	Running a series of Mif2Go conversions	990
36.5.7	Including runfm in a multi-step or scheduled process	991
36.6	Comparing runfm with the DCL command-line filter	991
36.7	Operating runfm across a network	992
36.8	Using runfm for other FrameMaker plug-ins	993
37	Converting via DCL	995
37.1	How the DCL filter works	995
37.2	Using the DCL filter	996
37.2.1	Understanding where to run DCL	996
37.2.2	Preparing for conversion	996
37.2.3	Converting a single MIF or DCL file	996
37.2.4	Converting a group of MIF or DCL files	997
37.2.5	Merging ancillary Help files with DCL	997
37.3	DCL command-line syntax	998
37.3.1	Command-line switch -f <i>format</i>	998
37.3.2	Command-line switch -o <i>output</i>	999
37.3.3	Command-line argument <i>input</i> ...	999
37.3.4	Command-line switch -v	1000
37.3.5	Additional command-line switches	1000
37.4	Command-line examples	1000
37.4.1	Creating a document information file	1001
37.4.2	Writing converted files to a different directory	1001
37.4.3	Converting a group of files to RTF	1001
37.4.4	Converting a file to HTML	1001
37.4.5	Converting from one DCL format to another	1001
37.4.6	Generating DITA output via command line	1002
37.5	Converting in multiple steps via DCL	1002
37.6	Specifying output file paths and names	1002
37.7	About DCL technology	1003
37.7.1	DCL file structure	1003
37.7.2	Writing DCL conversion modules	1003
38	Generating intermediate output	1005
38.1	Producing MIF with Mif2Go vs. FrameMaker	1005
38.2	Generating MIF output	1006
38.2.1	Understanding how MIF files are generated	1006
38.2.2	Setting up a FrameMaker MIF project	1006

38.2.3	Specifying which files to include in MIF output	1007
38.2.4	Saving FrameMaker 8 files as FrameMaker 8 MIF.	1008
38.2.5	Saving FrameMaker 9+ files as FrameMaker 7 MIF.	1008
38.2.6	Specifying file extensions for MIF output.	1008
38.3	Converting to ASCII DCL	1009
38.3.1	Setting up an ASCII DCL project	1009
38.4	Generating ASCII DCL output.	1011
38.4.1	Including generated files in ASCII DCL output	1011
38.4.2	Specifying type and file extension for ASCII DCL output	1012
38.4.3	Exporting embedded graphics via ASCII DCL output	1012
A	WAI marker library for HTML	1013
B	Distribution files	1017
C	Document and conversion files	1019
D	Technical support for Mif2Go	1029
E	DITA <bookmeta> template	1039
F	Content model configuration	1043
	RTF keyword index	1047
	HTML/XML keyword index	1059
	Subject index	1085

1 Getting started with Mif2Go

Figure 1-1 Mif2Go conversion process62

2 Planning a conversion project

(No illustrations)

3 Converting a book or document

Figure 3-1 Choose Project dialog78

Figure 3-2 Import FrameMaker Template80

Figure 3-3 Convert variables to text.....80

Figure 3-4 Include generated files81

Figure 3-5 **Mif2Go** Export dialog83

4 Editing configuration files

(No illustrations)

5 Setting basic conversion options

(No illustrations)

6 Converting to print RTF

Figure 6-1 Set Up Print RTF Project146

7 Producing on-line Help

(No illustrations)

8 Generating WinHelp

Figure 8-1 Set Up WinHelp Project.....245

9 Generating Microsoft HTML Help

Figure 9-1 Set Up MS HTML Help Project.....298

Figure 9-2 HTML Help Workshop Project tab304

Figure 9-3 HTML Help Workshop Window Types.....304

10 Generating OmniHelp

Figure 10-1 Set Up OmniHelp Project.....346

11 Generating JavaHelp or Oracle Help

Figure 11-1 Set Up Java Help Project375

12 Generating Eclipse Help

Figure 12-1 Set Up Eclipse Help Project404

13 Converting to HTML/XHTML

Figure 13-1 Set Up HTML/XML Project.....426

Figure 13-2 RGB color 0099CC440

Figure 13-3 Made with **Mif2Go**.....452

14 Converting to generic XML*(No illustrations)***15 Converting to DITA XML**

Figure 15-1 Set Up DITA Project. 479

16 Configuring DITA maps*(No illustrations)***17 Converting to DocBook XML**

Figure 17-1 Set Up DocBook Project. 560

18 Splitting and extracting files

Figure 18-1 Splitting a file 585

Figure 18-2 Extracting a file. 585

19 Creating HTML links*(No illustrations)***20 Providing navigation in HTML**

Figure 20-1 Positions of files in TechGuide.book 643

21 Mapping text formats to HTML/XML*(No illustrations)***22 Setting up CSS for HTML**

Figure 22-1 CSS set-up options 683

23 Including graphics in HTML*(No illustrations)***24 Converting tables to HTML***(No illustrations)***25 Generating WAI markup for HTML**

Figure 25-1 TableSummary marker 762

26 Identifying HTML table structure for WAI*(No illustrations)***27 Marking HTML table cells for WAI**

Figure 27-1 Extent of row and column groups. 780

Figure 27-2 Extent of column and row spans. 782

Figure 27-3 Extent of column and row IDs 783

28 Working with macros*(No illustrations)***29 Working with FrameMaker markers***(No illustrations)***30 Working with templates***(No illustrations)*

31 Working with graphics	
Figure 31-1 FrameMaker 7+ <i>Object Attributes</i> dialog.	898
32 Working with content models	
<i>(No illustrations)</i>	
33 Overriding configuration settings	
<i>(No illustrations)</i>	
34 Automating Mif2Go conversions	
Figure 34-1 Defining character format Popup	934
Figure 34-2 Defining a character format for pop-up hotspots	934
Figure 34-3 Pop-up hotspot in WinHelp	934
Figure 34-4 <i>Edit User Variable</i> dialog.	943
35 Producing deliverable results	
<i>(No illustrations)</i>	
36 Converting via runfm	
<i>(No illustrations)</i>	
37 Converting via DCL	
<i>(No illustrations)</i>	
38 Generating intermediate output	
Figure 38-1 Set Up FrameMaker MIF Project	1007
Figure 38-2 Set Up ASCII DCL Project	1010
A WAI marker library for HTML	
Figure A-1 Primary travel method in near future.	1015
B Distribution files	
<i>(No illustrations)</i>	
C Document and conversion files	
<i>(No illustrations)</i>	
D Technical support for Mif2Go	
<i>(No illustrations)</i>	
E DITA <bookmeta> template	
<i>(No illustrations)</i>	
F Content model configuration	
<i>(No illustrations)</i>	

About this guide

Table 0-1 Mif2Go User's Guide formats and archives.	41
---	----

1 Getting started with Mif2Go

(No tables)

2 Planning a conversion project

(No tables)

3 Converting a book or document

Table 3-1 General set-up options and settings	81
---	----

Table 3-2 Mif2Go export options and configuration settings	84
--	----

4 Editing configuration files

Table 4-1: Absolute vs. relative file-path settings	106
---	-----

5 Setting basic conversion options

Table 5-1 Output types, file extensions, project configuration files	110
--	-----

Table 5-2 Basic graphic conversion options for HTML/XML	127
---	-----

Table 5-3 Basic graphic conversion options for RTF	128
--	-----

6 Converting to print RTF

Table 6-1 Print RTF set-up options and configuration settings	146
---	-----

Table 6-2 RTF differences between Word 7/95 and later versions	150
--	-----

Table 6-3 Default font types and metrics for RTF	167
--	-----

Table 6-4 RTF font types and font families	168
--	-----

Table 6-5 Effects of cross-reference settings in Word	176
---	-----

Table 6-6 Graphics scale percentages for Word versions	191
--	-----

7 Producing on-line Help

Table 7-1 Index link options for KeywordRefs in HTML-based Help	216
---	-----

Table 7-2 Effects of drop-down format properties	229
--	-----

8 Generating WinHelp

Table 8-1 WinHelp set-up options and configuration settings	245
---	-----

Table 8-2 Starting and following format properties for topics and hotspots	269
--	-----

Table 8-3 Effects of format properties on topics and hotspots	269
---	-----

9 Generating Microsoft HTML Help

Table 9-1 HTML Help set-up options and configuration settings	299
---	-----

Table 9-2 ALink and KLink jump properties for HTML Help	311
Table 9-3 Binary TOC/Index advantages and disadvantages for HTML Help ...	321
Table 9-4 Map and alias files needed for CSH in HTML Help	328
Table 9-5 Rationale for HHW settings by CHM role	340
Table 9-6 HTML Help Workshop settings for stand-alone vs. merged CHMs ...	340

10 Generating OmniHelp

Table 10-1 OmniHelp viewer control files included in the distribution	344
Table 10-2 OmniHelp data and control files generated by Mif2Go	345
Table 10-3 OmniHelp set-up options and configuration settings	346
Table 10-4 OmniHelp navigation features	356
Table 10-5 OmniHelp viewer files copied from OHViewPath to WrapPath	370

11 Generating JavaHelp or Oracle Help

Table 11-1 JavaHelp set-up options and configuration settings	375
Table 11-2 [JavaHelpOptions] pop-up and secondary window properties	396
Table 11-3 [<i>JavaHelp window name</i>] window-access object properties	396
Table 11-4 Oracle Help for Java window properties	398

12 Generating Eclipse Help

Table 12-1 Eclipse Help set-up options and configuration settings	405
Table 12-2: Eclipse Help properties in either MANIFEST.MF or plugin.xml ...	407

13 Converting to HTML/XHTML

Table 13-1 HTML and XHTML set-up options and configuration settings	426
Table 13-2 Color numbers for default FrameMaker colors	439
Table 13-3 Ways to express Web-safe RGB color values	440
Table 13-4 FrameMaker color conversion anomaly	441
Table 13-5 Default options for Confluence 4.x XHTML	450
Table 13-6 Characters replaced or removed for W3C validation	454

14 Converting to generic XML

(No tables)

15 Converting to DITA XML

Table 15-1 DITA set-up options and configuration settings	479
Table 15-2 Precedence of DITA topic type assignment methods	524
Table 15-3 Predefined marker types for DITA XML	536

16 Configuring DITA maps

Table 16-1 DITA map navigation elements from custom markers	548
Table 16-2 Roles of component files in a bookmap	551
Table 16-3 Components for bookmap frontmatter and backmatter	553

Table 16-4	Predefined marker types for DITA maps and bookmarks.	556
17 Converting to DocBook XML		
Table 17-1	DocBook set-up options and configuration settings	561
Table 17-2	Predefined marker types for DocBook	583
18 Splitting and extracting files		
Table 18-1	Precedence of HTML page titles	595
Table 18-2	Extract code insertion methods	598
Table 18-3	Basic macro-insertion keywords and locations	599
Table 18-4	Keyword prefixes for split or extract code insertion	599
Table 18-5	Code insertion keywords for split and extract files	600
Table 18-6	Predefined macro variables for splits and extracts	601
Table 18-7	Predefined marker types for extracts	602
Table 18-8	Predefined macro variables for extract replacement code.	603
19 Creating HTML links		
<i>(No tables)</i>		
20 Providing navigation in HTML		
Table 20-1	Indirect navigation macros for files in a book	637
Table 20-2	Equivalent browse macros and variables by file position	637
Table 20-3	Default destination and label values for browse macros	637
Table 20-4	Component macro variables for browse macros	638
Table 20-5	Scope of [NavigationMacros] keywords	642
Table 20-6	Default values of text-link browse keywords	642
Table 20-7	Default values of button browse keywords	642
Table 20-8	Values of variables in navigation links for TechGuide.book	643
21 Mapping text formats to HTML/XML		
Table 21-1	HTML properties for paragraph and character formats.	654
Table 21-2	Special characters to replace for HTML/XML output.	661
22 Setting up CSS for HTML		
Table 22-1	Default CSS file options when [HtmlOptions]Stylesheet is used	687
Table 22-2	CSS-dependent default values of options	688
23 Including graphics in HTML		
<i>(No tables)</i>		
24 Converting tables to HTML		
Table 24-1	Precedence of table and cell property assignment methods.	728
Table 24-2	Browser-dependent HTML tags for tables	731
Table 24-3	Default counts of table header rows/columns and footer rows	734

25 Generating WAI markup for HTML

(No tables)

26 Identifying HTML table structure for WAI

Table 26-1	Format properties for WAI table-cell attributes	768
Table 26-2	Using paragraph formats to identify table cells (example)	770

27 Marking HTML table cells for WAI

Table 27-1	WAI scope attributes for table cells	775
Table 27-2	WAI id/header table cell attributes	778
Table 27-3	ColGroup property effects	785
Table 27-4	RowGroup property effects	786

28 Working with macros

Table 28-1	Predefined macros for HTML output	792
Table 28-2	Character literals for macro variables	798
Table 28-3	Predefined macro variables	800
Table 28-4	Operators for HTML macro expressions	812
Table 28-5	Format components for displaying expression results	814
Table 28-6	Predefined control-structure elements	815
Table 28-7	String operators in macro expressions	817
Table 28-8	Macro code placement properties	823

29 Working with FrameMaker markers

Table 29-1	Custom marker types with predefined effects	832
Table 29-2	Elements to which attribute markers apply, by output type	835
Table 29-3	Effects of [MarkerTypes] properties	839

30 Working with templates

Table 30-1	Output-type-specific general configuration files	850
Table 30-2	Configuration chain for Mif2Go User's Guide title page	855
Table 30-3	Intended scope of settings by configuration type	857
Table 30-4	Chain of general configuration files for HTML Help output	858
Table 30-5	Output types and starting project configuration files	859
Table 30-6	Editable local output-specific configuration files	860
Table 30-7	Macro configuration files	861
Table 30-8	Configuration options determined at run time	863
Table 30-9	Template flag values for importing formats	864

31 Working with graphics

Table 31-1	RTF replacement graphics file mappings and locations	892
------------	--	-----

32 Working with content models

Table 32-1 Configuration files for Mif2Go built-in content models	906
--	-----

33 Overriding configuration settings

Table 33-1 Precedence of settings in configuration files and templates	920
Table 33-2 Fixed-key configuration sections subject to overrides.	925
Table 33-3 Text configuration sections subject to overrides	926
Table 33-4 Cross-reference sections subject to overrides	928
Table 33-5 HTML table sections subject to overrides	928
Table 33-6 HTML graphic sections subject to overrides.	930

34 Automating Mif2Go conversions

(No tables)

35 Producing deliverable results

Table 35-1 Default files copied from project directory to wrap directory	963
Table 35-2 Files copied by default to the wrap directory.	964
Table 35-3 Default graphics files copied for assembly	966
Table 35-4 Automation settings activated by CompileHelp or FTSCCommand . . .	972
Table 35-5 Default base file name for deliverables archive.	975

36 Converting via runfm

Table 36-1 Command-line options for runfm	981
--	-----

37 Converting via DCL

Table 37-1 DCL intermediate input and output options.	1000
---	------

38 Generating intermediate output

Table 38-1 FrameMaker MIF set-up options and configuration settings.	1007
Table 38-2 ASCII DCL set-up options and configuration settings	1011

A WAI marker library for HTML

Table A-1 Special marker types for WAI table attributes	1013
Table A-2 Examples of WAI table markers.	1014
Table A-3 Special marker types for WAI graphic attributes	1015
Table A-4 Special marker types for WAI link attributes	1016

B Distribution files

Table B-1 Mif2Go distribution files	1017
--	------

C Document and conversion files

Table C-1 Location of document files	1019
Table C-2 Location of conversion files	1019

D Technical support for Mif2Go

Table D-1 Examples of build numbers for **Mif2Go** DLL files 1034

E DITA <bookmeta> template

(No tables)

F Content model configuration

(No tables)

About this guide

The **Mif2Go User's Guide** describes how to install and use Omni Systems **Mif2Go** version 4.1 software and configuration files, to convert FrameMaker documents to any of several output types. This guide assumes you are familiar with FrameMaker, and also with the output type to which you are converting FrameMaker files.

In this section:

§ [Availability](#) on page 41

§ [New information](#) on page 41

§ [Colophon](#) on page 50

Availability

The **Mif2Go User's Guide** is available in the formats listed in [Table 0-1](#). All editions except the FrameMaker source files and the PDF edition are produced with **Mif2Go**.

Table 0-1 Mif2Go User's Guide formats and archives

Format	Archive	Starting file	Comments*
DITA XML	UGMif2Go_dita55.zip	<i>Not applicable</i>	DITA version 1.1, including bookmap; not validated
DocBook XML	UGMif2Go_dbk55.zip	<i>Not applicable</i>	DocBook version 4.5; not validated
Eclipse Help	UGMif2Go_EH55.zip	<i>Not applicable</i>	Requires Eclipse platform or infocenter
FrameMaker	UGfrm55.zip	<i>Not applicable</i>	FrameMaker, configuration, template, graphics files
HTML	UGMif2Go_HTM55.zip	_ugmif2go.htm	Requires JavaScript to enable CSS
HTML Help	UGMif2Go_HH55.zip	ugmif2go.chm	Includes context-sensitive Help for Mif2Go Must be registered for network use
JavaHelp 2	UGMif2Go_JH55.zip	ugmif2go.jar	Requires Java Runtime Environment
OmniHelp	UGMif2Go_OH55.zip	_ugmif2go.htm	Requires browser and JavaScript
Oracle Help	UGMif2Go_OHJ55.zip	ugmif2go.hs	Requires Java Virtual Machine
PDF	UGMif2Go_PDF55.zip	ugmif2go.pdf	Designed for duplex printing
Word 2007	UGMif2Go_RTF55.zip	ugmif2go.rtf	Includes active cross references and hypertext links
XHTML	UGMif2Go_XH55.zip	_ugmif2go.htm	Requires JavaScript for CSS; some browsers ignore

*See § [Colophon](#) on page 50 for additional information.

You can download any of these archives from Omni Systems:

<http://mif2go.com/>

Extract the files from their archive before you try to view them.

New information

The **Mif2Go User's Guide** is a perpetual work-in-progress, largely unreviewed. This section identifies substantive additions and corrections since prior editions. Corrections are shown in **red**.

May 15, 2013 version 4.1, update 55

Getting started with Mif2Go:

- **Copy m2rbook.dll and m2gframe.dll to the correct Plugins directory.** These files are located in %OMSYSHOME%\m2g\plugin in your distribution, and they must be copied to your FrameMaker \fminit\Plugins directory; see §1.3.3.2 [Finish installing Mif2Go](#) on page 57.

Converting a book or document:

- **Access the Mif2Go Configuration Manager from the Choose Project dialog.** New button; see §3.3 [Creating a Mif2Go conversion project](#) on page 78.

Editing configuration files:

- **Edit settings with the Mif2Go Configuration Manager.** New tool, accessible from the *Choose Project* dialog; see §4.2 [Editing files with the Configuration Manager](#) on page 91.
- **Use relative path settings with care!** Relativity of some path settings has changed, notably those listed in [Table 4-1](#); see §4.5 [Specifying file paths in configuration settings](#) on page 105.
- **A relative path specified for [Automation]CopyAfterFrom is relative to the project directory (*correction*).** Not to the wrap directory; see §4.5 [Specifying file paths in configuration settings](#) on page 105.
- **A relative path specified for [Setup]TemplateFileName is relative to the location of the configuration file in which the setting occurs (*correction*).** Not to the source directory; see §4.5 [Specifying file paths in configuration settings](#) on page 105.

Setting basic conversion options:

- **One setting for output file extension.** Use [Setup]FileSuffix for all output types; see §5.1.1 [Checking output type and file extension](#) on page 110. Though still recognized, the following are deprecated in favor of [Setup]FileSuffix, which overrides them in any event:
 WordperfectSuffix
 WordSuffix
 XMLSuffix
 HTMLSuffix
- **Turn off warnings about uncatalogued formats.** New setting ShowUndefinedFormats, on by default; see §5.2 [Logging conversion events](#) on page 115.
- **Identify and use Structured FrameMaker attributes.** New settings IDAttrName and IDRefAttrName, new section [AttributeMarkers]; see §5.8 [Converting structured documents](#) on page 135.

Converting to print RTF:

- **Double any backslashes in included RTF code.** And double them again in a macro; see §6.15 [Including RTF code for Word output](#) on page 194.
- **Remove unwanted empty paragraphs at end of topics.** New setting FrameEndPara; see §6.10.9 [Omitting content from RTF output](#) on page 174.

Producing on-line Help:

- **TopicAlias markers for context-sensitive help.** Instead of hypertext newlink markers. New custom marker type TopicAlias; see §7.10.2 [Specifying CSH mappings](#) on page 241.

Generating Microsoft HTML Help:

- **Specify binary TOC or index for HTML Help.** Newly documented settings **BinaryTOC** and **BinaryIndex**; see §9.9.2 [Choosing whether to generate binary contents or index](#) on page 320.
- **Exclude selected topics from FTS for HTML Help.** An inelegant hack via new custom marker **Search**; see §9.11 [Providing full-text search \(FTS\) for HTML Help](#) on page 326.
- **Generate a CSH map file.** New settings **MakeCshMapFile**, **CshMapFileNumStart**, and **CshMapFileNumIncrement**; see §9.12.3 [Specifying and generating a map file for CSH links](#) on page 329.

Generating OmniHelp:

- **Make compound terms searchable in OmniHelp.** Newly documented setting **CompoundWordChars**; see §10.10.3 [Making compound terms searchable](#) on page 362.
- **Exclude content from full-text search.** New custom marker **Search**; see §10.10.7 [Excluding content from being searched](#) on page 363.

Converting to HTML/XHTML:

- **Show FrameMaker condition indicators in HTML output.** New section **[ConditionOptions]**; new settings **UseConditionalFlagging**, **CSSFlagsFile**, **WriteFlagsFile**, and **ReferenceFlagsFile**. See §13.10.3 [Displaying condition indicators in HTML with CSS](#) on page 447.
- **Prevent adjacent HTML <pre> elements from merging.** New setting **MergePre**; see §13.6.5 [Preventing adjacent <pre> elements from merging](#) on page 438.

Converting to DITA XML:

- **Wrap anchored images in <fig> as an exception.** Newly documented HTML format property **Figure**; see §15.7.2 [Specifying what to include in a <fig> wrapper](#) on page 517.
- **Keep selected topics out of the TOC.** New marker type **DITANoTOC**; see §15.9.6 [Omitting a DITA topic from the TOC](#) on page 527.
- **CSH targets via TopicAlias markers are included by default.** New setting **UseTopicAlias**; see §15.14 [Including CSH targets in DITA XML](#) on page 535.

Configuring DITA maps:

- **Omit id attribute from chapter maps.** New setting **UseMapID**; see §16.2.1.6 [Specifying the ID for a ditamap](#) on page 542.

Creating HTML links:

- **Tell Mif2Go to leave link text case alone.** Although its internal default value is **No**, **MakeFileHrefsLower** is set to **Yes** in system file **d2htm_config.ini**, which is referenced for every HTML output type. If you want **Mif2Go** to leave case alone in hypertext links, you must explicitly set **MakeFileHrefsLower** to **No** in a project or local configuration file; see §19.2.6 [Forcing link text to lowercase](#) on page 613.

Mapping text formats to HTML/XML:

- **Assign HTML tags in new configuration sections.** **[ParaTags]** and **[CharTags]** replace **[ParaStyles]** and **[CharStyles]**, respectively; see:
 - §21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646
 - §21.4 [Mapping character formats](#) on page 653
- **Distinguish paragraph (block) from character (inline) format properties.** New sections **[HTMLParaStyles]** and **[HTMLCharStyles]** supersede **[HTMLStyles]**, which is still honored; see §21.5 [Assigning properties to text formats](#) on page 653.

- **Exclude size attribute from font tags.** Newly documented setting `UseFontSize`; see §21.7.6 [Excluding face and size attributes from font tags](#) on page 666.

Setting up CSS for HTML:

- **Omit element tags from CSS selectors.** New setting `SelectorIncludesTag`; see §22.7.10 [Omitting tags from CSS selectors](#) on page 696.
- **HTML table rows are now wrapped in groups by default.** Reversed setting `HeadFootBodyTags`; see §24.3.2.4 [Wrapping table row groups](#) on page 732.

Working with macros:

- **Distinguish between paragraph and character code inclusions.** New sections:

<u>Old section</u>	<u>New for paragraph formats</u>	<u>New for character formats</u>
[StyleCodeAfter]	[ParaStyleCodeAfter]	[CharStyleCodeAfter]
[StyleCodeBefore]	[ParaStyleCodeBefore]	[CharStyleCodeBefore]
[StyleCodeEnd]	[ParaStyleCodeEnd]	[CharStyleCodeEnd]
[StyleCodeReplace]	[ParaStyleCodeReplace]	[CharStyleCodeReplace]
[StyleCodeStart]	[ParaStyleCodeStart]	[CharStyleCodeStart]

The old sections are deprecated, but still honored; see §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

Working with FrameMaker markers:

- **Replace marker content with code.** New section `[MarkerTypeCodeReplace]`; see §29.7.2 [Surrounding marker content with code](#) on page 843.
- **No need to remap TopicAlias markers for CSH.** No longer required, and not advised for DITA output; see §29.3.2 [Understanding when to remap marker types](#) on page 837.

Converting via runfm:

- **"Save As PDF" via runfm.** New command-line option `-pdfsave` can avoid printer issues and TimeSavers issues; see §36.4.3.3 [Configuring PDF output: runfm -pdfsave option](#) on page 986.

May 1, 2012, version 4.0, update 54

Getting started with Mif2Go:

- **No "quick start" this time!** Omni Systems has massively reorganized the **Mif2Go** distribution; see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54.
- **System-wide configuration file.** Specify in one place any settings that apply to all projects; see §1.3.6 [Establish system-wide configuration settings](#) on page 58.
- **All Mif2Go executables are now referenced from the new Omni Systems directory structure,** *not* from the Windows system directory. See §1.4.2 [Update your Mif2Go installation](#) on page 61.

Planning a conversion project:

- **Produce a single output file from a FrameMaker book.** You really need a script to do this right; see §2.5.6 [Producing a single output file from a FrameMaker book](#) on page 73.

Editing configuration files:

- **New names for project configuration files!** Each is now named for its output type; see §5.1.1 [Checking output type and file extension](#) on page 110.
- **Keep at least one line of header text.** Each configuration file must start with a header line; see §4.4 [Understanding the rules for configuration settings](#) on page 102.

- **No more than one space after a *key=value* equals sign.** Do not try to line up values vertically with extra spaces *after* the equals (*before* is usually OK, though not always); see §4.4 [Understanding the rules for configuration settings](#) on page 102.
- **Comment out entire configuration sections with a single semicolon.** New tip; see §4.7 [Commenting out configuration sections](#) on page 107.
- **Append log files to a history file.** New setting `HistoryFileName`; see §5.2 [Logging conversion events](#) on page 115.
- **ObjectIDs change on show/hide conditional text.** Always! So use carefully; see §5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118.
- **Condition show/hide settings might not work in FrameMaker 10+.** Go back to importing FrameMaker templates instead; see §5.4.1 [Applying condition Show/Hide settings](#) on page 123.
- **Include a suffix in generated equation file names.** New setting `EqSuffix`; see §5.9.4 [Providing a file-name suffix for equations](#) on page 137.

Converting to print RTF:

- **Specify a Windows code page for print RTF output.** New setting `CodePage`; see §6.2.4 [Specifying the default output language and code page](#) on page 147.
- **Keep trailing tabs in Word output.** New setting `TrailingTabs`; see §6.8.3 [Altering tab behavior for Word output](#) on page 164.
- **Produce .doc or .docx files via Word macro and a `SystemEndCommand`.** See §6.17.1 [Supporting more than one version of Word](#) on page 195.

Producing on-line Help:

- **Produce index meta elements for Microsoft Help Viewer.** New setting `UseHVIndex`, new custom marker type `HVIndex`; see §7.5.2 [Preparing index entries for Microsoft Help Viewer](#) on page 211.
- **Provide a sort order for Japanese index entries.** Most likely you will need help; see §7.5.9.5 [Defining Japanese index sort order](#) on page 218.

Generating WinHelp:

- **Get WinHelp viewers from Microsoft.** And so must every user; see §8.1 [Obtaining tools for WinHelp](#) on page 243
- **Tell Mif2Go where to find the WinHelp compiler.** Previously undocumented setting `Compiler`; see §8.2.13 [Compiling a WinHelp project](#) on page 250.
- **Specify copyright and date for WinHelp *Version Information*.** Previously undocumented settings `HelpCopyright` and `HelpCopyDate`; see §8.2.13 [Compiling a WinHelp project](#) on page 250.
- **Insert a separator between text and footnotes in WinHelp.** New setting `FootnoteSeparator`; see §8.3.7 [Converting footnotes](#) on page 258.
- **Specify table width when you use column-width percentages for WinHelp.** New setting `TblFullWidth`; see §8.5.2 [Adjusting table appearance](#) on page 261.

Generating Microsoft HTML Help:

- **Omit code-page mapping for uncompiled HTML Help.** New setting `UseCodePage`; see §9.3.6 [Deciding whether to compile HTML Help](#) on page 300.
- **Generate Asian or Cyrillic HTML Help.** New code-page DLLs; see §9.13 [Generating HTML Help in non-Western languages](#) on page 331.
- **Fixed spaces become ideographs for Japanese HTML Help.** Or you can map them to something else; see §9.13 [Generating HTML Help in non-Western languages](#) on page 331.
- **Tell Mif2Go where to find the HTML Help compiler.** New setting `Compiler`; see §9.14.1 [Directing Mif2Go to run the HTML Help compiler](#) on page 333.

- **Compile HTML Help in a different locale.** Use command-line utility SBAppLocale; see §9.14.3 [Compiling in a different language](#) on page 335.

Generating OmniHelp:

- **.Include relative paths in OmniHelp subprojects to be merged.** See §10.12.2 [Listing and mapping OmniHelp subprojects](#) on page 367.

Generating JavaHelp or Oracle Help:

- **Specify options for Oracle Help in their own eponymous section.** New section [OracleHelpOptions]; for example, see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.
- **Add type attribute to list tags for JavaHelp and Oracle Help.** New CSS setting UseListTypeAttribute; see:
 - §11.3.9 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 384
 - §21.12.2.7 [Including or excluding the type list attribute](#) on page 678.

Converting to HTML/XHTML:

- **For ePub, XHTML output can provide input to Calibre.** See §13.1 [Deciding which type of output to produce](#) on page 424.
- **For HTML 5 output, set DOCTYPE as appropriate.** See §13.4.1 [Specifying HTML/XML version, DOCTYPE, and DTD](#) on page 429.
- **Omit generator information from HTML output (*not advisable*).** Previously undocumented setting [HtmlOptions]GeneratorTag=None; see §13.4.4 [Including or omitting HTML/XML generator information](#) on page 433.
- **Generate XHTML for Confluence 4.x.** New settings Confluence, ConfluenceLinks, and ConfluenceLink*; see §13.12 [Generating XHTML for Confluence 4.x](#) on page 449.
- **Supply hover text for terms and acronyms in HTML.** New format property GlossTitle, new sections [GlossTitles] and [GlossFiles]; see §13.11 [Providing hover text for terms in HTML](#) on page 448.
- **Turn on UseSpacers explicitly to indent tables and figures!** Important if your output relies on this setting, because *the default value has been changed to No*. See:
 - §23.6.3 [Indenting images](#) on page 716
 - §24.5.1 [Indenting tables](#) on page 747.

Converting to generic XML:

- **Keep Shift+Enter line breaks.** New setting UseXMLbr; see §14.4.5 [Configuring forced returns for XML](#) on page 465.

Converting to DITA XML:

- **Map formats to elements depending on topic type.** New functionality in [DITAAliases], new predefined macro variable \$\$_ditastart; see §15.4.3.8 [Mapping paragraph format aliases algorithmically](#) on page 491.
- **Replace change bars or overlines with tags for DITA XML output.** When UseTypographicElements=Yes, you get <chbar> and <over> elements, respectively; see §15.4.4.2 [Including typographic elements in addition to mapped formats](#) on page 493.
- **Give every DITA element that can accommodate it an ID.** New setting SetElementIDs; see §15.4.6.2 [Including an id attribute in every element](#) on page 496.
- **Specify attributes for the root element of a topic.** New section [DITATopicRootAttrs], new marker type DITATopicRootAttrs; see §15.4.6.3 [Specifying attribute values for the root element of a topic](#) on page 497.

- **Include an `outputclass` attribute in the DITA XML root element.** New marker type `DITATopicOutputclass`; see §15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498.
- **Avoid constructing invalid ancestries.** New setting `UseCommonNames`; see §15.5.6 [Avoiding invalid ancestries](#) on page 504.
- **Configure nested lists for DITA XML.** See §15.5.8 [Configuring nested lists](#) on page 505.
- **Use relative or absolute table widths in DITA XML.** New setting `TableColsRelative`; see §15.6.5.3 [Specifying relative vs. absolute widths for table columns](#) on page 513.
- **Give table footer rows an `@outputclass` of their own.** New settings `UseTableFooterClass`, `TableFooterClass`; see §15.6.2 [Marking table footer rows for future reference](#) on page 511.
- **Specify placement of figure titles for DITA.** New setting `FigureTitleStartsFigure`; see §15.7.2 [Specifying what to include in a <fig> wrapper](#) on page 517.
- **Omit size attributes from images for DITA.** Newly applicable setting `[Graphics]GraphScale`; see §15.7.3 [Omitting size attributes from images for DITA output](#) on page 518.
- **Include `MathFullForm` equation objects in `<alt>` text.** New setting `MathFullForm`; see §15.7.5 [Including MathFullForm equations in <alt> elements](#) on page 518.
- **Include `FrameMaker` DPI values in `<image>` attributes.** New setting `UseOtherpropsDPI`; see §15.7.6 [Including the original image DPI as an attribute](#) on page 518.
- **Rename DITA topic files to make the file names readable.** `FrameScript` method; see §15.8.3.2 [Renaming DITA topic files with FrameScript](#) on page 521.
- **Specify an `outputclass` attribute for cross-reference wrappers.** New settings `XrefWrapClass`, `FootnoteWrapClass`, and `IndexWrapClass`; see §15.10.2 [Specifying an outputclass for cross-reference wrappers](#) on page 528.

Configuring DITA maps:

- **Adjust map levels for missing heading levels.** See §16.2.3 [Accounting for missing topic levels](#) on page 544.
- **Include DITA bookmark `<appendix>` elements in `<part>`.** New setting `AllowPartAppendix`; see §16.3.4 [Extending <part> to include <appendix>](#) on page 550.
- **Exclude book-level reltable from a bookmark.** New setting `MapBookRelTable`; see §16.3.6 [Excluding the book-level reltable from a bookmark](#) on page 550.
- **Include multiple TOCs, indexes, other booklist items in a bookmark.** New sections `[IndexMarkerOutputClass]`, `[DITABookmapOutputclasses]`; see §16.4.3 [Including multiple booklist components of the same type](#) on page 553.
- **Accommodate placeholders for extra bookmark elements.** New sections `[BookmarkElementBefore]`, `[BookmarkElementAfter]`; see §16.4.7 [Including placeholders for additional bookmark elements](#) on page 555.

Converting to DocBook XML:

- **Allow multiple images in a figure element for DocBook.** Newly applicable setting `MultiImageFigures`; see §17.7.1 [Deciding what to include in a figure element](#) on page 581.
- **Include additional content in figure elements for DocBook.** Newly applicable setting `CloseFigAfterImage`; see §17.7.1 [Deciding what to include in a figure element](#) on page 581.

- **Specify placement of figure titles for DocBook.** New setting `FigureTitleStartsFigure`; see §17.7.1 [Deciding what to include in a figure element](#) on page 581.
- **Omit size attributes from images for DocBook.** Newly applicable setting `[Graphics]GraphScale`; see §17.7.3 [Omitting size attributes from images for DocBook](#) on page 582.

Splitting and extracting files:

- **Insert space or a separator between HTML topics in a single output file.** New `[Inserts]` keyword `TopicBreak`; see:
 - §18.5.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 599
 - §28.9.2 [Invoking macros at predetermined points in output](#) on page 821.

Creating HTML links:

- **Replace spaces in links with hyphens or underscores.** Instead of only alphanumerics for `XrefSpaceChar` or `HyperSpaceChar`; see §19.2.5 [Replacing problem characters in links](#) on page 612.

Mapping text formats to HTML/XML:

- **Prevent line breaks and preserve leading spaces at the paragraph format level.** New `[HTMLParaStyles]` format property `NoWrap`; see §21.3.6 [Stripping paragraph properties](#) on page 650.
- **Font tags are now excluded from HTML output by default.** The default values for `NoFonts` and `NoSymbolFont` have been reversed; see:
 - §21.7.4 [Including or excluding font tags](#) on page 665
 - §21.7.5 [Managing font tags for symbol fonts](#) on page 666
- **Replace change bars or overlines with tags for HTML/XML output.** Use new “pseudo tag” names `chbar` and `over`; see §21.8.2 [Choosing how to treat typographic elements](#) on page 667.

Setting up CSS for HTML:

- **Replace spaces in CSS class names with hyphens or underscores.** Instead of only alphanumerics as a value for `ClassSpaceChar`; see §22.7.1 [Understanding CSS class name restrictions](#) on page 691.
- **Flag alternate-language glyphs with CSS classes.** New setting `UseCharRangeClasses`, new section `[CharacterRangeClasses]`; see §22.7.6 [Assigning CSS classes based on Unicode character ranges](#) on page 694.

Including graphics in HTML:

- **HTML image indent spacer default has been reversed.** If you rely on this deprecated technique, you must explicitly set `UseSpacers=Yes`; see §23.6.3 [Indenting images](#) on page 716.
- **Omit empty `alt` attribute values from HTML output.** New setting `AllowEmptyAlt`; see §23.8 [Providing \(or omitting\) alternate text for images](#) on page 718.

Converting tables to HTML:

- **HTML table indent spacer default is reversed.** If you rely on this deprecated technique, you must explicitly set `UseSpacers=Yes`; see §24.5.1 [Indenting tables](#) on page 747.
- **Strip HTML code from selected tables.** With configuration macros or `Config` markers; see:
 - §24.6.4 [Turning processing on and off around selected tables](#) on page 750
 - §24.7.2 [Removing table-specific tags from selected tables](#) on page 754.

Working with macros:

- **Add a visible trailing space to a macro.** New convention “\~” for trailing spaces; see §28.1.1.3 [Escaping special characters in macro definitions](#) on page 789.
- **Predefined macro variables for project name and path.** New variable `$$_prjname`; also, variable `$$_prjpath` can now be used in macros as well as in system commands. See [Table 28-3](#) on page 800.
- **More ways to manipulate strings.** New macro expression operators **lower**, **upper**, and **replace with**; see [Table 28-4 Operators for HTML macro expressions](#) on page 812.
- **Pass a parameter to a macro, and capture its value.** New predefined macro variable `$$_macroparam`; see §28.7 [Passing a parameter to a macro](#) on page 820.

Working with FrameMaker markers:

- **Treat marker content as plain text for XML output.** New marker property type **Text**; see §29.7.3 [Processing marker content as text for XML/HTML/XHTML](#) on page 844.

Working with templates:

- **Configuration templates play a major role now.** New **[Templates]** section with new settings **Document**, **Configs**, and **Macros**; see §30.1 [Working with configuration templates](#) on page 849.
- **[FDK]ConfigTemplate is deprecated in favor of [Templates]Configs.** Replace all instances of the former; see §30.2 [Referencing configuration files and templates](#) on page 851.
- **Include one or more templates for document-specific settings.** New setting **[Templates]Document**; see §30.3 [Including document-specific configuration files](#) on page 852.

Working with graphics:

- **Extract embedded graphics imported from Word.** Without knowing their formats; see §31.2.7 [Exporting embedded graphics imported from Word](#) on page 886.

Automating Mif2Go conversions:

- **System-command keywords have new names, new configuration section:**

<u>Old</u>	<u>New</u>
[BookFileCommands]	[Automation]
BookStartCommand	SystemStartCommand
BookWrapCommand	SystemWrapCommand
BookEndCommand	SystemEndCommand
FileStartCommand	SystemStartCommand
FileWrapCommand	SystemWrapCommand
FileEndCommand	SystemEndCommand

See §34.4.1 [Specifying system commands](#) on page 938.

- **Whip FrameMaker variables into shape for use in file names.** Whack spaces and punctuation with macros; see §34.8.4.2 [Including FrameMaker variables in output file names](#) on page 949.
- **Use variable `<$$_objectid>` in file names.** To guarantee uniqueness, if you are determined to name output files yourself; see §34.8.4.5 [Specifying a file-name prefix or suffix](#) on page 950.

Producing deliverable results:

- **Copy ancillary files before and after conversion.** New **[Automation]** settings **CopyBeforeFrom**, **CopyBeforeFiles**, **CopyAfterFrom**, **CopyAfterFiles**; see: §35.5 [Gathering additional files before converting](#) on page 960

§35.6.6 [Listing extracurricular files to put in the wrap directory](#) on page 964.

- **Default value of wrapPath is now ._wrap, relative to the project directory;** see §35.6.1 [Specifying a wrap directory](#) on page 961.
- **Gather referenced graphics files for distribution.** New setting `CopyOriginalGraphics`; see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.
- **Default value of ShipPath is now .._ship, relative to the project directory;** see §35.12.1 [Specifying a shipping directory for deliverables](#) on page 975.

Converting via runfm:

- **Log in as administrator before you set up runfm.** Because you must write to the Windows Registry; see §36.2 [Setting up FrameMaker for unattended operation](#) on page 980.

Colophon

PDF edition The PDF edition of the **Mif2Go User's Guide** is:

- intended for duplex printing (using both sides of the paper).
- sized to fit either US Letter or A4 size paper.

Word edition The Microsoft Word edition of the **Mif2Go User's Guide** consists of separate RTF chapter files optimized for Word 2000, with active cross references and hypertext links. To make this edition viewable in other versions of Word, see §6.17.1 [Supporting more than one version of Word](#) on page 195.

HTML and XHTML editions The OmniHelp edition of the **Mif2Go User's Guide** has been tested with the following browsers:

- Internet Explorer 9.x
- Mozilla Firefox 16.x
- Opera 12.x
- Google Chrome 26.x

The XHTML edition works poorly with Mozilla browsers when viewed locally; furthermore, these browsers ignore CSS entirely when the XHTML edition is viewed on the Web.

HTML and XHTML editions have an overview TOC on the opening page, a detailed TOC next, and several indexes at the end. These navigation features were created in FrameMaker and implemented with settings in **Mif2Go** configuration files. The navigation tables and breadcrumb links at top and bottom of each text page were produced automatically with **Mif2Go** macros.

DITA and DocBook editions The DITA and DocBook editions have not been validated. These editions are provided primarily as demonstrations; their structure has not been made to conform to architectural conventions or best practices.

Source files The **Mif2Go User's Guide** is an unstructured FrameMaker version 8.0 document,. The FrameMaker source files in `UGMif2Go_frm55.zip` include path information. You can download this archive from Omni Systems:
<http://mif2go.com>

(5/18/13 13:09:57)

1 Getting started with Mif2Go

This section tells you how to install **Mif2Go**, and how to update **Mif2Go**. Topics include:

- §1.1 [What you need to know](#) on page 51
- §1.2 [What you need to have](#) on page 53
- §1.3 [What you need to do](#) on page 54
- §1.4 [How to update Mif2Go](#) on page 61
- §1.5 [How Mif2Go works](#) on page 62
- §1.6 [How to start and stop Mif2Go](#) on page 63
- §1.7 [How to work with Mif2Go](#) on page 63
- §1.8 [How to uninstall Mif2Go](#) on page 64

1.1 What you need to know

To use **Mif2Go** effectively, it is best if all the following apply:

- You are intimately acquainted with the structure of your FrameMaker document.
- You understand the output type to which you are converting your document.
- You have a good idea which FrameMaker features and formats you want to map to which output features.

Conversions to a final format such as HTML or RTF are easy, and you can often wing it, achieving acceptable output with default options in just a few minutes. But conversions to another source format, such as DocBook or DITA XML, can be Very Hard. There are no shortcuts. You might need days or weeks to get it right, working with small test documents, before you can go into production.

In this section:

- §1.1.1 [How Mif2Go is organized](#) on page 51
- §1.1.2 [File, directory, and path names](#) on page 51
- §1.1.3 [Output types you can specify](#) on page 52
- §1.1.4 [Languages and character sets](#) on page 53

1.1.1 How Mif2Go is organized

To convert your FrameMaker source to an HTML or RTF representation, **Mif2Go** provides the means to perform two primary tasks:

- Map FrameMaker formats to output formats.
- Define presentational properties of those output formats.

Mif2Go also carries out a number of output-type-dependent secondary tasks, such as constructing Help file infrastructure.

To map FrameMaker formats to output formats, **Mif2Go** relies on both rules and instance mark-up. Rules come from settings in configuration files; instance mark-up is in the FrameMaker files themselves, in the form of custom FrameMaker markers.

1.1.2 File, directory, and path names

This section describes naming conventions that apply to all the resources in your conversion project.

No spaces or punctuation in file or path names

Book files, chapter files, graphics files, and any other files referenced by your FrameMaker document, or by configuration settings, should have names and paths that conform to the following guidelines:

- No spaces in file or path names (because FrameMaker does not always handle them correctly).
- Only alphanumeric characters in file or path names (even underscores can cause problems on some systems).
- Maximum 256 characters in a file name, 1,024 characters in a path name (Windows limits).
- At most one period in a file or path name.

Mif2Go is not the problem!

Although **Mif2Go** supports file and path names that include underscores and spaces, the output type you choose, or the target operating system where your output will be deployed, might not. For example, Microsoft acknowledges a known defect in HTML Help when you use underscores in file names. Some flavors of UNIX do not like underscores, either. In the interest of compatibility, we advise against file names that are not strictly alphanumeric.

Single period followed by extension

Use only one period in a path or file name, followed by the correct extension for the type of file. Many Windows programs break because this tiny character is misused. It is best not to challenge the easily confused software on your computer.

1.1.3 Output types you can specify

Choose from the following output types:

<u>RTF / WinHelp</u>	<u>HTML / XML</u>	<u>HTML-based Help</u>	<u>Other formats</u>
Word 7/95	Standard HTML	MS HTML Help	ASCII DCL
Word 8/97+	XHTML	Eclipse Help	FrameMaker MIF
WinHelp 4/95	Generic XML	JavaHelp	
WordPerfect	DITA XML	Oracle Help for Java	
	DocBook XML	OmniHelp	

Print RTF

Mif2Go handles styles, tables, and graphics. You can convert to Word or WordPerfect. See §6 [Converting to print RTF](#) on page 141 for more information.

WinHelp RTF

You can configure and generate WinHelp RTF with **Mif2Go**; and if you have access to Help Workshop (unfortunately no longer available from Microsoft) you can compile WinHelp files that need no further tweaking. **Mif2Go** produces all the files you need, including a TOC, an Index, and a Help project file. See:

§8 [Generating WinHelp](#) on page 243.

HTML-based Help

You can use **Mif2Go** to configure and generate several flavors of HTML-based Help. **Mif2Go** produces all the files you need, typically including TOC, Index, and a Help project file. For the two Java formats, **Mif2Go** prepares the map file; for HTML Help, the aliases file. See:

§9 [Generating Microsoft HTML Help](#) on page 295

§10 [Generating OmniHelp](#) on page 341

§11 [Generating JavaHelp or Oracle Help](#) on page 373

§12 [Generating Eclipse Help](#) on page 403.

HTML and XML

Mif2Go can produce HTML 4.01, XHTML 1.0, and XML 1.0 files from your FrameMaker document, and also create Cascading Style Sheets. You can include arbitrary JavaScript anywhere in HTML output. You can produce ePub input from **Mif2Go** XHTML output. See:

	<p>§13 Converting to HTML/XHTML on page 423</p> <p>§14 Converting to generic XML on page 457.</p>
DITA and DocBook	<p>Mif2Go can produce DITA XML and DocBook XML from DITA documents. See:</p> <p>§15 Converting to DITA XML on page 473</p> <p>§17 Converting to DocBook XML on page 557.</p>
Intermediate format	<p>You can run a Mif2Go conversion in two stages, stopping the first part of the process when DCL files have been created. You can use the intermediate files for other purposes, or modify them and then run Mif2Go again to continue the conversion. See §38 Generating intermediate output on page 1005 for more information.</p>

1.1.4 Languages and character sets

In addition to Western languages, **Mif2Go** supports Russian, Greek, and Central/Eastern European languages. For HTML/XML outputs **Mif2Go** supports *all* languages, via Unicode (UTF-8). For RTF outputs, **Mif2Go** supports only single-byte languages, although it is possible to produce decent Japanese RTF for Word.

Mif2Go does not currently support non-Unicode double-byte languages, nor right-to-left languages such as Hebrew and Arabic. However, if you use **Mif2Go** to produce Microsoft HTML Help, you can specify Japanese, Chinese, or Korean output, via Asian code pages that you must download separately; see §1.2 [What you need to have](#) on page 53.

1.2 What you need to have

To use **Mif2Go** your computer system should be equipped as follows:

- Windows: 2000, XP, Vista, or 7
- Intel-compatible Pentium-level processor
- 1+ GB memory recommended

In addition to **Mif2Go**, you will need at least some of the following software:

[Adobe FrameMaker](#)
[Text editor](#)
[File comparison tool \(optional\)](#)
[Archiving tool](#)
[Ancillary tools for Help output.](#)

Adobe FrameMaker	<p>You will need Adobe FrameMaker version 5.5.6 or a later version. Mif2Go supports FrameMaker versions 9 and 10, with the following exceptions:</p> <ul style="list-style-type: none"> • Books containing books are not supported; nested books are ignored. • Non-FrameMaker files in books are not supported; such files are ignored.
Text editor	<p>To work with configuration files you will need a text editor, such as Notepad, that uses ANSI or UTF-8 encoding; do not use UTF-16.</p>
File comparison tool (optional)	<p>If you modify any of the local configuration files provided with your Mif2Go distribution, you might find it helpful to have a file comparison tool such as WinMerge; see §1.3.8 Obtain a file comparison tool (optional) on page 60.</p>
Archiving tool	<p>To have Mif2Go automatically archive the output from your conversion projects for safe storage or for distribution, you will need an archiving program that can be run from a Windows command line, such as WinZip (via <code>wzzip</code>) or PKZip.</p>

*Ancillary tools for
Help output*

If you intend to generate on-line Help, you will need one or more of the following tools; see §1.3.5 [Obtain tools for Help systems or eBooks](#) on page 58:

WinHelp	Microsoft Help Workshop (<i>no longer available</i>) and viewer
HTML Help	Microsoft HTML Help Workshop
JavaHelp	Java Runtime Environment (JRE) and JavaHelp software
Oracle Help for Java	Oracle Help for Java Developer's Kit 2.0
Eclipse Help	Java Runtime Environment (JRE), Eclipse Platform

To produce Asian code-page output, such as for HTML Help in Japanese, you will also need two enormous ICU DLLs: `icudt40.dll` (13MB) and `icuuc40.dll` (1MB). These DLLs are available in archive `icu401.zip` (6 MB), which you can download from the Omni Systems Web site. These DLLs are not needed for RTF output.

1.3 What you need to do

Starting with version 4.0 of **Mif2Go**, you must establish a new Omni Systems directory structure for **Mif2Go** files. Follow the instructions in this section the first time you install a version of **Mif2Go** later than version 3.3. To update **Mif2Go** version 4.0 or a later version, see §1.4 [How to update Mif2Go](#) on page 61.

In this section:

- §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54
- §1.3.2 [Download a Mif2Go distribution](#) on page 56
- §1.3.3 [Install Mif2Go](#) on page 56
- §1.3.4 [Make Omni Systems executables accessible](#) on page 57
- §1.3.5 [Obtain tools for Help systems or eBooks](#) on page 58
- §1.3.6 [Establish system-wide configuration settings](#) on page 58
- §1.3.7 [Locate document-specific settings](#) on page 60
- §1.3.8 [Obtain a file comparison tool \(optional\)](#) on page 60
- §1.3.9 [Download the Mif2Go User's Guide \(optional\)](#) on page 61

1.3.1 Set up a framework for Omni Systems applications

If this is the first Omni Systems application to be installed on your system, you must establish a new directory structure for executables, configuration templates, and ancillary files. You must:

- [Create an Omni Systems home directory](#)
- [Create an Omni Systems environment variable](#)
- [Create FrameMaker environment variables](#)
- [Verify that your new framework is accessible.](#)

*Create an Omni
Systems home
directory*

Unless you already have the Omni Systems directory structure in place, create a new directory on your system for all Omni Systems applications; for example, `D:\omsys`. This is your Omni Systems home directory.

Do not place the Omni Systems home directory:

- on a network drive; latency issues can cause intermittent problems
- on any path that contains spaces.

See §D.2.5 [Check path names, file names, and drive location](#) on page 1032.

Create an Omni
Systems
environment
variable

Unless your system already has system environment variable %OMSYSHOME% that specifies an absolute path to the Omni Systems home directory, you will need to create this variable.

1. In *Control Panel* (on Windows XP, for example):

Control Panel > System > Advanced > Environment Variables

2. If OMSYSHOME is not listed in the **System variables** section, click **New** to create this environment variable. For example:

Variable name: OMSYSHOME

Variable value: D:\omsys

Click **OK**.

3. Under **System variables** select **Path** and click **Edit**. You should see something like:

Variable name: Path

Variable value: C:\a\long\string;C:\of\directory\paths;

4. Place your cursor in the **Variable value** field and press the End key on your keyboard, to navigate to the end of the Path value.

5. *If the last character in the Path value is a semicolon*, very carefully add the following to the *end* of the Path value:

%omshome%\common\bin;

Otherwise, *if the last character in the Path value is not a semicolon*, very carefully add the following to the *end* of the Path value:

;%omshome%\common\bin

Be sure to include the leading semicolon!

6. Click **OK** three times to save the environment variable definition and revised system path, and return to *Control Panel*. Now **Mif2Go** will be able to find all the Omni Systems files.

Create
FrameMaker
environment
variables

Unless your system already has one or more environment variables of the form %FMnHOME% that specify the path(s) to your FrameMaker installation(s), you will need to create a variable for each version of FrameMaker installed on your system.

For example, if you have FrameMaker versions 7.x, 8.x, and 10.x, you will need %FM7HOME%, %FM8HOME%, and %FM10HOME%. To see if environment variable %FM7HOME% (for example) is already present, in a Windows *Command Prompt* window, type:

```
set fm7home
```

and press *Enter*. If this environment variable is already defined, Windows will display its definition.

To define %FM7HOME% (for example) on Windows XP (for example), go to:

Control Panel > System > Advanced > Environment Variables

If FM7HOME is not listed in the **System Variables** section, click **New** to create this environment variable. For example:

Variable name: FM7HOME

Variable value: "C:\Program Files\Adobe\FrameMaker7.0"

Note: The double quotes are *essential* if the path contains spaces.

Click **OK** three times to save the definition and return to *Control Panel*.

Now **Mif2Go** will be able to find the version of FrameMaker you are currently using.

Verify that your new framework is accessible

Reboot your Windows system. Then open a command-prompt window, type **dc1**, and press **Enter**. You should see a usage message for **dc1.exe**. If you see a “not found” message instead, something is wrong.

Next: §1.3.2 [Download a Mif2Go distribution](#) on page 56.

1.3.2 Download a Mif2Go distribution

Register (or log in) at:

<http://mif2go.com>

and go to one of the **Download** pages.

Download *one* of the following, depending on what Omni Systems software is already present on your system:

Already on your system:

Mif2Go, any version

DITA2Go but not **Mif2Go**

Neither **DITA2Go** nor **Mif2Go** version 4.0 or later

What to download:

m2g_update_55.zip

m2g_addon_55.zip

m2g_full_55.zip

Next: §1.3.3 [Install Mif2Go](#) on page 56.

1.3.3 Install Mif2Go

Before you begin:

- If you do not yet have an Omni Systems home directory on your system, first §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54.
- If you already have **Mif2Go** version 4.0 or later on your system, skip this section and instead §1.4.2 [Update your Mif2Go installation](#) on page 61.

Why no installer?

Omni Systems does not provide an installer for **Mif2Go**. This is for transparency; you *know* **Mif2Go** does not “call home”, make changes to the Windows Registry, put files where your company policy does not permit them, nor alter any other files on your system. Information-system technicians can see exactly what will happen, and adjust the instructions as needed to comply with company policy.

In this section:

§1.3.3.1 [Extract files from the Mif2Go distribution archive](#) on page 56

§1.3.3.2 [Finish installing Mif2Go](#) on page 57

1.3.3.1 Extract files from the Mif2Go distribution archive

To install **Mif2Go** for the first time, place **Mif2Go** distribution **m2g_full_55.zip** in your Omni Systems home directory and extract all files, allowing the extraction process to create subdirectories.

Check extraction

If your unzip or uncompress utility puts the extracted files in a directory named after the distribution archive, move them up one level so they are directly under the Omni Systems home directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54).

For example, executables should be here:

%OMSYSHOME%\common\bin

not here:

%OMSYSHOME%\m2g_full_55\common\bin

Check directory
structure

You should have a directory structure that looks like this:

```
%OMSYSHOME%
|
|--common
|   |--bin
|   |--local
|   +--system
|
+--demo
|   +--Mif2Go_Demo
|
+--m2g
|   |--local
|   |--plugin
|   |--system
|   |--usersguide
|   +--zip
```

1.3.3.2 Finish installing Mif2Go

To complete the installation:

1. Move the **Mif2Go** distribution .zip file to subdirectory %OMSYSHOME%\m2g\zip, where it will be available for future reference.
2. On your desktop, create a shortcut to %OMSYSHOME%\common\bin\m2gcm.exe. This gives you double-click access to the **Mif2Go Configuration Manager**, which you can use to edit project settings.
3. In Windows Explorer, navigate to:

```
%OMSYSHOME%\m2g\usersguide
```

Right-click ugmif2go.chm, select **Properties**, and click **Unblock**. This is a standard Microsoft “security” measure, used for all CHM files downloaded from the Internet, or contained in archives downloaded from the Internet.

4. Double-click ugmif2go.chm to register the **Mif2Go** context-sensitive Help system with Windows, so **Mif2Go** can find it.
5. On your desktop, create a shortcut to:

```
%OMSYSHOME%\m2g\usersguide\ugmif2go.chm
```

This gives you access to the **Mif2Go User’s Guide**, HTML Help edition. You can download other editions from Omni Systems; see [Availability](#) on page 41.

6. Close FrameMaker, then copy the following files from %OMSYSHOME%\m2g\plugin to your FrameMaker fminit\plugins directory:


```
m2rbook.dll
m2gframe.dll
```
7. If you are using a version of FrameMaker earlier than version 9, delete file .cache from your FrameMaker fminit\plugins directory.

Next: §1.3.4 [Make Omni Systems executables accessible](#) on page 57.

1.3.4 Make Omni Systems executables accessible

Starting with **Mif2Go** version 4.0, all Omni Systems executables except the FrameMaker plug-ins are located in the following directory:

```
%omsysHOME%\common\bin
```


Make sure you add this directory to your system PATH; see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54.

Old Mif2Go files Delete from your Windows system directory (\windows\system32 or, for 64-bit systems, \windows\SysWOW64) the following **Mif2Go** components, for which there are new versions:

```
dcl.exe
drmf.dll
dwrtd.dll
dwhm.dll
dwinf.dll
libexpat.dll
```

Note: If you leave the old files in the system directory, Windows will use them instead of your new executables, and you will wonder why **Mif2Go** does not work as expected.

Automated build systems If you have been using automated build systems for **Mif2Go** that rely on finding executables in the Windows system directory, you have two choices:

- Change your scripts to access %omshome%\common\bin instead; *this is the preferred method.*
- Every time you update **Mif2Go**, copy all the new executables from %omshome%\common\bin to the Windows system directory.

Next, if you plan to produce on-line Help or ebook formats, §1.3.5 [Obtain tools for Help systems or eBooks](#) on page 58; otherwise, §1.3.6 [Establish system-wide configuration settings](#) on page 58.

1.3.5 Obtain tools for Help systems or eBooks

If you plan to generate any of the Help formats, you will need additional tools to compile or complete your Help project.

MS HTML Help Tools, including HTML Help Workshop, are in the Microsoft Library:
<http://msdn.microsoft.com/en-us/library/ms669985.aspx>

If you plan to generate HTML Help in non-Western languages, you will also need the ICU library; see §9.13 [Generating HTML Help in non-Western languages](#) on page 331.

Oracle Help Oracle Help for Java, available from Oracle:
<http://www.oracle.com/technetwork/topics/ohj50ext-089966.html>

JavaHelp JavaHelp, available from java.net:
<http://download.java.net/javadesktop/javahelp/>

Eclipse Help Eclipse SDK or Infocenter, available from Eclipse:
<http://www.eclipse.org/downloads/>

WinHelp Microsoft Help Workshop for 32-bit WinHelp is no longer available; however, you can still obtain the viewer; see §8.1 [Obtaining tools for WinHelp](#) on page 243.

eBooks You can produce ePub input with **Mif2Go** XHTML output, then use free converter Calibre: <http://calibre-ebook.com>

Next, §1.3.6 [Establish system-wide configuration settings](#) on page 58.

1.3.6 Establish system-wide configuration settings

To specify values for system-wide settings that apply to all Omni Systems applications, open the following configuration file in a text editor:


```
%omshome%\common\local\config\local_omshome.ini
```

This configuration file is accessed by all other **Mif2Go** configuration files, through chains of links; the settings it contains are available to all **Mif2Go** projects. You can place in this file any setting that will apply to most or all of your **DITA2Go** and **Mif2Go** projects. For particular FrameMaker documents or conversion projects, you can override these settings in output- or document-specific configuration files.

Note: For proper syntax, see §4.4 [Understanding the rules for configuration settings](#) on page 102.

Specify settings for any of the following features that might be required for your conversion projects:

[Archiving program and options](#)

[HTML Help compiler command](#)

[Eclipse Help zip command](#)

[JavaHelp, Oracle Help index and JAR commands](#)

[WinHelp copyright statement and compiler command](#)

*Archiving
program and
options*

Mif2Go can package the output from your conversion projects in .zip files for distribution. You must provide an archiving program that can be run from a command line, and specify appropriate parameters:

```
[Automation]
ArchiveCommand = path\to\archiver
ArchiveStartParams = parameters preceding name of archive file
ArchiveEndParams = parameters following name of archive file
ArchiveExt = file extension; usually zip
MoveArchive = Yes to move archive, No to copy archive
LogAuto = Yes to record archiving steps in the run log
```

See §35.11 [Archiving deliverables](#) on page 973. The starting and ending parameters for the archive command have default values; for some output types you will need to override these defaults in your project configuration file.

*HTML Help
compiler
command*

For HTML Help projects, **Mif2Go** can run the Microsoft HTML Help compiler for you, to produce compiled CHM files. Unless the compiler is on your system PATH, you must tell **Mif2Go** where to find it:

```
[MSHtmlHelpOptions]
Compiler = path\to\hhc
```

See §9.14.1 [Directing Mif2Go to run the HTML Help compiler](#) on page 333.

*Eclipse Help zip
command*

For Eclipse Help projects, **Mif2Go** can package your HTML topic files in doc.zip. You must provide an archiving program that can be run from a command line, and specify appropriate parameters:

```
[EclipseHelpOptions]
ZipCommand = path\to\archiver
ZipParams = all required parameters
```

See §12.8 [Packaging Eclipse Help files](#) on page 419.

*JavaHelp, Oracle
Help index and
JAR commands*

For JavaHelp projects, **Mif2Go** can run the JavaHelp indexer to produce a full-text search index, and also package the output in a .jar file. You must specify the indexer and JAR commands:

```
[JavaHelpOptions]
FTSCommand = path\to\jhindexer
JarCommand = path\to\jar
```

For Oracle Help projects, **Mif2Go** can run the Oracle Help indexer to produce a full-text search index:

```
[OracleHelpOptions]
FTSCommand = java -mx256m oracle.help.tools.index.Indexer
```

See §11.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 387, and §11.6.1 [Creating a JAR file](#) on page 391.

*WinHelp
copyright
statement and
compiler
command*

For WinHelp projects, **Mif2Go** can provide a copyright statement to be included in the WinHelp .hlp project file:

```
[HelpOptions]
HelpCopyright = your copyright statement
```

Mif2Go can also run the Microsoft WinHelp compiler for you, to produce compiled HLP files. Unless the WinHelp compiler is already on your system PATH, you must tell **Mif2Go** where to find it. For example:

```
[HelpOptions]
; Compiler = path\to\hwc; can include run parameters
Compiler = hwc /c /e
```

See §8.2.13 [Compiling a WinHelp project](#) on page 250.

1.3.7 Locate document-specific settings

The default location for document-specific configuration files is a directory named `_config`, located parallel to your project directory. To change this location, or to specify different default names for these files, you must edit values of the following options in configuration file `%OMSYSHOME%\m2g\local\config\local_m2g_config.ini`:

```
[Setup]
; Used when creating document-specific configuration files:
; LocalConfigPath = location relative to project directory
LocalConfigPath = ..\_config\
; WinHelpDocName = default document-specific .ini for WinHelp output
WinHelpDocName = winhelp_doc.ini
; WordDocName = default document-specific .ini for Word output
WordDocName = word_doc.ini
; HTMLDocName = default document-specific .ini for all HTML outputs
HTMLDocName = html_doc.ini
```

To determine whether the default location is appropriate, see §30.3.3 [Deciding where to keep document-specific configuration files](#) on page 854.

Your **Mif2Go** installation is now complete, and you can set up a **Mif2Go** conversion project, or run an existing conversion. Your existing project configuration files will work without modification. See §2 [Planning a conversion project](#) on page 65.

1.3.8 Obtain a file comparison tool (optional)

If you do not already have software on your system that compares text files and allows you to accept or reject changes, consider downloading WinMerge for this purpose:

<http://winmerge.org/>

If you customize local copies of any of the configuration templates or macro libraries included in **Mif2Go** distributions, with WinMerge you will be able to see what has changed when you update the system copies.

1.3.9 Download the Mif2Go User's Guide (optional)

The HTML Help edition of the **Mif2Go User's Guide** is included with your **Mif2Go** installation. You can download other editions of the **Mif2Go User's Guide**, in several formats, from the Omni Systems Web site:

<http://mif2go.com>

Log in, and go to **Download > User's Guide**. Download any or all current editions.

1.4 How to update Mif2Go

If you have never installed **Mif2Go** on your system, or if you have a version of **Mif2Go** earlier than version 4.0, you must first §1.3.3 [Install Mif2Go](#) on page 56. Then in future you can use one of the update procedures described here.

In this section:

§1.4.1 [Change from the evaluation version to a licensed version](#) on page 61

§1.4.2 [Update your Mif2Go installation](#) on page 61

§1.4.3 [Try out Mif2Go beta executables](#) on page 62

1.4.1 Change from the evaluation version to a licensed version

If you have only the evaluation version, you can order a full license for **Mif2Go**. Go to:

<http://mif2go.com>

Log in, and go to **Orders > Order Mif2Go**. Follow the instructions there.

If your evaluation version is earlier than version 4.0, you must install **Mif2Go** from scratch; first, §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54.

Next, §1.4.2 [Update your Mif2Go installation](#) on page 61.

1.4.2 Update your Mif2Go installation

As long as you subscribe to **Mif2Go** support, you are automatically notified of new versions of **Mif2Go**.

Note: Updates do not change anything in your %OMSYSHOME%\m2g\local or %OMSYSHOME%\common\local subdirectories, nor replace executables that are unchanged since the prior update.

To update **Mif2Go** to the latest version:

1. Download m2g_update_55.zip into your Omni Systems home directory, %OMSYSHOME%.
2. Extract all files from m2g_update_55.zip, allowing old files to be overwritten.
3. **Only if you are using build systems that require executables in the system directory**, copy all files from %omsyshome%\common\bin into system directory \windows\system32 (on 64-bit systems, \windows\SysWOW64), overwriting any **DITA2Go** or **Mif2Go** components already there. *You will have to do this every time you update Mif2Go.* A much better plan would be to change your build scripts to reference %omsyshome%\common\bin instead.
4. Close FrameMaker, then copy the following files from %OMSYSHOME%\common\plugin to your FrameMaker fminit\plugins directory:
m2rbook.dll

m2gframe.dll

5. If you are using a version of FrameMaker earlier than version 9, delete file .cache from your FrameMaker fminit\plugins directory.
6. Consult %omshome%\m2g\userguide\history.txt and [New information](#) on page 41 to see what has changed since your last update.

1.4.3 Try out Mif2Go beta executables

You can obtain the latest beta revisions of **Mif2Go** distribution files dr*.dll and dw*.dll from the Omni Systems Web site:

<http://mif2go.com>

Log in, go to **Download > Registered Software**, and scroll down to **Beta Components**. Each .dll file is in an individual .zip archive with a name that includes the build number, such as mwrtf372.zip.

To see if a .dll file you have is the latest revision:

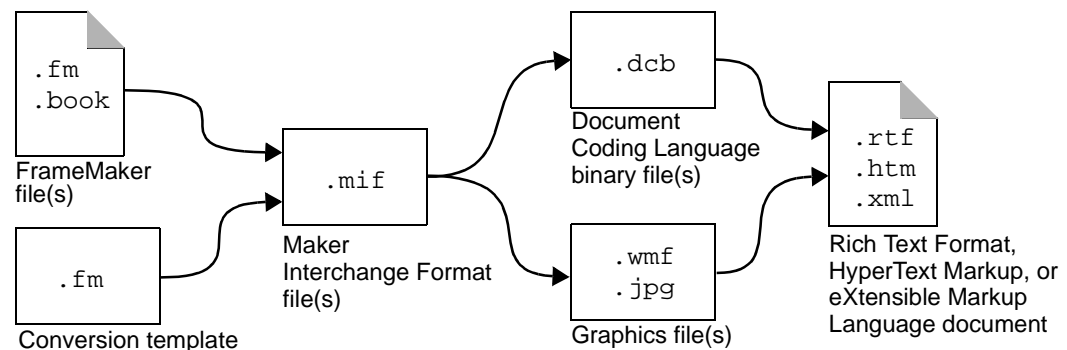
1. In Windows Explorer, right-click the file icon for the copy of the .dll file that is located in %OMSYSHOME%\common\bin.
2. Go to **Properties > Version**.
3. Compare the build number displayed after **File version:** with the build number that appears in the name of the corresponding .zip archive. The larger number represents the later version.

Extract the beta .dll file to %OMSYSHOME%\common\bin (or to your Windows system directory, if you are using build systems that require executables to reside in the system directory).

1.5 How Mif2Go works

To convert FrameMaker documents, **Mif2Go** first saves the document files as MIF (Maker Interchange Format) files, then passes the MIF through an input filter to produce intermediate DCL (Document Coding Language) files, then passes the DCL files through an output filter to produce the final output type. [Figure 1-1](#) shows the main features of this process. **Mif2Go** deletes the intermediate DCL files after completing the conversion.

Figure 1-1 Mif2Go conversion process



File placement

Mif2Go places a project file and a FileID file in the same directory as your FrameMaker document, and a project configuration file and other conversion files in whatever directory you specify for the output; see §C.2 [Identifying conversion files](#) on page 1020.

If you subsequently move your document, project, and FileID files to another directory, **Mif2Go** still expects to find the FileID file in the original location; this is because the configuration file contains an absolute (full path) reference to the location of the FileID file at the time the project was created. *If **Mif2Go** does not find a FileID file in the original directory, **Mif2Go** creates a new FileID file in that directory.*

*Conversion
template*

If you use a FrameMaker conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67) and experiment with conversion settings, you can produce output that needs no further editing, without altering your original FrameMaker document.

*Evaluation
version*

The evaluation version of **Mif2Go** works identically to the shipping version, with one small exception: the evaluation version randomly modifies about 10% of the text in your document to ensure that it is used for evaluation only, per the evaluation license. No changes are made to the Windows Registry, and there are no time or size limits. Graphics are not modified at all.

1.6 How to start and stop Mif2Go

*Starting **Mif2Go***

1. If you have not yet set up your **Mif2Go** project: with your FrameMaker book or document file open, on the FrameMaker **File** menu click **Set Up Mif2Go Export...** to specify a name, location, and output type for the project; §3.3 [Creating a Mif2Go conversion project](#) on page 78.
2. On the FrameMaker **File** menu, click **Save Using Mif2Go...** to select the project and start the conversion (see §3.6 [Converting documents](#) on page 82).

*Stopping **Mif2Go***

How to cancel a **Mif2Go** conversion before it finishes depends on which version of Windows you are using and whether you are running **Mif2Go** from the command line or from FrameMaker. Try one of the following:

- Press **Ctrl+C**, which works in some Windows environments.
- Open the Windows Task Manager (on Windows NT/2000/XP/Vista/7) or press **Ctrl+Alt+Del** to bring up the *Close Program* dialog (on Windows 9x/ME):
 1. Select the **Mif2Go** process, **dc1.exe**.
 2. Click **End Task**.

1.7 How to work with Mif2Go

The methodology for converting documents with **Mif2Go** is iterative:

1. Run your conversion project using the defaults for the output type you specify.
2. Look at the results, and pick one thing you want different.
3. Look up that feature in the **Mif2Go User's Guide** (this document), and find out what setting you need to change to get what you want.
4. Make that setting in your project configuration file.
5. Rerun the conversion, then go to [Step 2](#).

What makes this feasible is the running time, which for a sample document set is about 3 seconds; and is seldom more than a few minutes for any size project. So you do not lose your train of thought waiting for completion.

1.8 How to uninstall Mif2Go

To uninstall **Mif2Go**, delete all files and subdirectories from directory %OMSYSHOME%\m2g. If you have no other Omni Systems applications installed, you can also delete the Omni Systems home directory and all subdirectories.

2 Planning a conversion project

This section describes useful things to know and do before you use **Mif2Go** to convert documents. Topics include:

- §2.1 [Naming files, directories, and paths](#) on page 65
- §2.2 [Naming FrameMaker formats](#) on page 66
- §2.3 [Understanding Mif2Go configuration files](#) on page 66
- §2.4 [Importing formats from a conversion template](#) on page 67
- §2.5 [Preparing documents for conversion](#) on page 69
- §2.6 [Establishing a conversion environment](#) on page 74
- §2.7 [Setting up multiple interlinked HTML projects](#) on page 75
- §2.8 [Preparing deliverables after conversion](#) on page 75

2.1 Naming files, directories, and paths

No spaces or punctuation in file or path names

Book files, chapter files, graphics files, and any other files referenced by your FrameMaker document (or by the configuration file) should have names and paths that conform to the following restrictions:

- No spaces in file or path names (because FrameMaker does not always handle them correctly).
- Only alphanumeric characters in file or path names (even underscores can cause problems).
- Maximum 256 characters in a file name, 1,024 characters in a path name (Windows limits).
- At most one period in a file or path name.

Mif2Go is not the problem!

Although **Mif2Go** supports file and path names that include underscores and spaces, the output type you choose, or the target operating system where your output will be deployed, might not. For example, Microsoft lists a known defect in HTML Help when you use underscores in file names. Some flavors of UNIX do not like underscores, either. Some Help outputs, notably JavaHelp and Eclipse Help, but even HTML Help, will fail if there are spaces in paths within the project, because the display software is UNIX-based and treats spaces as delimiters. In the interest of compatibility, we advise against file names and paths that are not strictly alphanumeric.

Spaces cause problems in FrameMaker

Spaces in file and path names cause obscure problems for many Windows applications, including FrameMaker itself. For example, if a referenced graphic is missing, the path name FrameMaker presents in the *Import* dialog ends at the first space. You might have no idea what file FrameMaker is looking for, and there is no workaround. For example, if you have files named `Image 200.gif`, `Image 201.gif`, and so on, the dialog would ask for `\somepath\Image`. All you could do is skip the missing graphics file, then look very carefully for gray boxes once the document is open. A graphics file is probably the very worst kind of file to give a name that contains spaces.

Note: *Presence of spaces in file or path names could invalidate requests for Mif2Go technical support.*

Single period followed by extension

Use only one period in a path or file name, followed by the correct extension for the type of file. Many Windows programs break because this tiny character is misused. It is best not to challenge the easily confused software on your computer.

<i>Book name must not duplicate a chapter name</i>	If you are converting a FrameMaker book, make sure the base name of the book file is different from the base name of any of the chapter files. This is especially important for generating OmniHelp or DITA XML, where output files with the same name could get overwritten; and for generating WinHelp, where duplicate names could cause unexpected results.
<i>No duplicate chapter names</i>	No two chapter files in a FrameMaker book may have the same base file name. Mif2Go does not support duplicate file names in the same book. If you are using Mif2Go to create on-line Help, make separate Help projects instead, each with only unique FrameMaker file names, and then merge the projects later.

2.2 Naming FrameMaker formats

Mif2Go does not impose restrictions on FrameMaker format names. However, your conversion projects might go more smoothly if you adhere to the following guidelines. Avoid using format names that:

- are identical for a paragraph format and a character format
- differ *only* in case (such as *Body* and *body*)
- contain punctuation or other non-alphanumeric characters.

<i>Paragraph and character format names should differ</i>	Using the same name for a paragraph format and a character format can cause problems, unless you are careful to specify properties for those formats in the correct configuration section; see §21.5 Assigning properties to text formats on page 653. Even then, if you use any characters in symbol fonts, those characters might be incorrectly mapped. Mif2Go writes a warning to the log file if your FrameMaker document uses the same name for both a character and a paragraph format.
<i>Do not rely on difference in case</i>	A common issue is using (for example) both <i>Bulleted</i> and <i>bulleted</i> as paragraph format names, with different definitions. This is valid in FrameMaker, <i>but nowhere else</i> , because most other programs—including browsers that process CSS, and Microsoft Word—do not consider a difference in case alone to constitute a difference in identity. These are not just issues for Mif2Go , but for the downstream software that must interpret the results of your conversion. Mif2Go writes a warning to the log file if your FrameMaker document uses format names that differ only in case.
<i>Escape reserved characters</i>	The characters “=”, “;”, and “[” must be escaped with a backslash when a format name is referenced in a Mif2Go configuration file.
<i>Eschew wildcard characters in names</i>	It is best not to use wildcard characters “*” and “?” in <i>any</i> names. However, you can turn off wildcard use (and turn on case sensitivity) if you need to accommodate formats with such names. See §5.1.7 Specifying how to treat cases, spaces, and wildcards on page 113 for more information.
<i>Spaces in format names can be a problem for HTML</i>	Spaces in format names are harmless if you are converting to RTF. If you are converting to HTML, spaces are a problem only if you have two format names that become the same when spaces are removed (which is required for CSS). For example, if your document uses both <i>heading 1</i> and <i>heading1</i> , there will be a conflict in the CSS file. Mif2Go writes a warning to the log file if your FrameMaker document uses format names that contain spaces, and removing the spaces results in duplicate names.

2.3 Understanding Mif2Go configuration files

To control the conversion process, **Mif2Go** uses conversion settings read from configuration files. A **Mif2Go** configuration file is an ASCII text file with extension `.ini`. A configuration file lists the options and settings that drive a particular conversion.

	To specify these options and settings, you must edit configuration files with a text editor, outside of FrameMaker; see §4 Editing configuration files on page 91.
<i>Project configuration file</i>	<p>The configuration file you work with most will most likely be your project configuration file, located in the project directory for your Mif2Go conversion project.</p> <p>As you work with Mif2Go, you might need any or all of the following additional configuration files, depending on the type and complexity of your projects:</p> <ul style="list-style-type: none"> Configuration template files Configuration files Mif2Go creates for you Configuration files from previous conversions Configuration files for individual chapters File identifier index
<i>Configuration template files</i>	Configuration template files hold settings that rarely change from project to project. Mif2Go maintains a hierarchy of configuration template files chained together; the chain is referenced from your project configuration file. Also, you can create one or more configuration templates of your own, or direct Mif2Go to use an existing configuration file as a template; see §30 Working with templates on page 849.
<i>Configuration files Mif2Go creates for you</i>	<p>For each new conversion project, Mif2Go creates a project configuration file for you, if a file of the same name is not already present in the project directory. Table 30-1 on page 850 lists the names of these starting configuration files. Each name begins with an underscore, to help you find the file in the project directory.</p> <p>If you put a project configuration file in the project directory before you run Mif2Go via FrameMaker, Mif2Go cannot make certain initial settings for you; see §1.5 How Mif2Go works on page 62. For this reason it is best to allow Mif2Go to create a <i>new</i> project configuration file when you start a new conversion project.</p>
<i>Configuration files from previous conversions</i>	<p>If you already have configuration files with the right settings for a particular document (perhaps a configuration file from a previous installation of Mif2Go or mif2rtf), you can place that configuration file in the project directory and Mif2Go will use it.</p> <p>Note: If you had Mif2Go version 3.3 on your system, each of your existing Mif2Go projects already has, in the project directory, a configuration file named either <code>mif2rtf.ini</code> (for RTF output) or <code>mif2htm.ini</code> (for HTML/XML output). Mif2Go version 4.0 uses those files if they are present, even though they do not have the newer file names.</p>
<i>Configuration files for individual chapters</i>	If you are converting a FrameMaker book, in special cases you might also need individual configuration files for certain chapter files; see §33.1 Using a different configuration for selected files on page 919.
<i>File identifier index</i>	For each FrameMaker book or single-file document, Mif2Go creates a FileID (“file identifier”) file in the same directory as your FrameMaker book or document, unless this file is already present. The FileID file is named <code>mif2go.ini</code> ; see §C.2.1 Conversion files created during set-up on page 1020.

2.4 Importing formats from a conversion template

For conversion purposes, **Mif2Go** can use FrameMaker **Import Formats** to temporarily apply a different FrameMaker template to your document. Applying formats from a template that more closely approximates what you want to see in a given type of output is the best and easiest first step to achieve successful single-sourcing from FrameMaker. Import is automatic, and affects only the MIF conversion files, not your original document files. If you have chapter files open when **Mif2Go** imports formats, those files remain

open, and they are not affected by the formats **Mif2Go** imports. You get true single-sourcing, and you have to specify changes only once, in FrameMaker.

To create a conversion template, prepare a new FrameMaker template that uses the same features and names as the original template for your document, but provides different definitions for the following:

- paragraph, character, table, and cross-reference formats
- conditions and variables
- master pages.

You can start just by saving a copy of your main FrameMaker template file under a different name, then change the definitions of only those formats that should look different in the output.

Note: If you are converting a FrameMaker book, and some chapter files in the book use a different template, you must create a separate configuration file for each of those chapter files, and an alternate conversion template; then you can direct **Mif2Go** to apply the alternate conversion template to those files. See §33.1 [Using a different configuration for selected files](#) on page 919.

If you plan to convert FrameMaker-generated files, such as contents, index, list of tables, and list of figures, do either of the following:

- import, into your main conversion template, all paragraph and character formats from all generated files you plan to convert; or,
- provide a separate configuration file and conversion template for each of those generated files.

Some of the changes you might want to make with a conversion template are as follows:

- Provide cross-reference formats that are underlined and in a different color, and that eliminate page numbers, for Help systems or for HTML output.
- Define TOC paragraph formats that do not include page numbers, for Help systems or for HTML output.
- Change the Show/Hide settings for conditional text, to substitute simpler versions of complex objects that have no counterpart in the output type, or to provide alternate text, or different illustrations. **Mif2Go** excludes from the output any hidden conditional text.
- Redefine paragraph formats to eliminate sideheads and tabs.
- Redefine heading formats to omit, add, or alter Frame Above or Frame Below.
- Reduce the size range of headings; for example, sizes larger than 18 pt or smaller than 10 pt often look bad in Help systems.
- Substitute redesigned master pages, to alter or eliminate headers, footers, or other background objects.

Mif2Go applies the conversion template as part of the conversion process, *without altering your FrameMaker document*. However, if you are using **Mif2Go** to produce an HTML output type and you specify CSS, your CSS settings override display properties both from the original FrameMaker file and from any imported template.

See also:

§3.4.1 [Importing formats from a FrameMaker template](#) on page 79

§34.1.4 [Importing formats and conditional text settings](#) on page 936

2.5 Preparing documents for conversion

You can improve your chances of success with **Mif2Go** by observing certain conventions in your FrameMaker documents, to make the result look the way you want in the output. Also, you can create a special conversion template in FrameMaker to implement these conventions on the fly.

In this section:

- §2.5.1 [Updating your document in FrameMaker](#) on page 69
- §2.5.2 [Planning for graphics processing](#) on page 69
- §2.5.3 [Replacing embedded graphics with referenced graphics](#) on page 69
- §2.5.4 [Setting up cross references to and from text insets](#) on page 70
- §2.5.5 [Creating hotspots for hypertext links](#) on page 72
- §2.5.6 [Producing a single output file from a FrameMaker book](#) on page 73
- §2.5.7 [Preparing a structured document for conversion](#) on page 73

See also:

- §8.2.3 [Deciding where to locate configuration settings](#) on page 245
- §13.2.3 [Preparing a document for conversion to HTML or XHTML](#) on page 426

2.5.1 Updating your document in FrameMaker

Mif2Go works reliably only on FrameMaker documents that have been updated and are free from unresolved cross references:

- If your document contains cross references or other links, make sure they are all resolved in FrameMaker before you use **Mif2Go**.
- If you intend to convert generated files, make sure they are updated first in FrameMaker.

If your FrameMaker document was originally converted from Microsoft Word via FrameMaker native Word import (not recommended), the document might contain a very large number of Word-generated cross-reference markers. By default, **Mif2Go** ignores these markers. If you have actually used any of these markers in FrameMaker, see §5.1.10 [Preserving Word-generated cross-reference markers](#) on page 114.

2.5.2 Planning for graphics processing

You will need to provide versions in the following formats of all imported-by-reference or copied-in graphics in your document, possibly via conditional-text settings applied with a conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67):

- BMP or WMF for Word or WinHelp
- JPEG, GIF, or PNG for HTML.

While **Mif2Go** can convert a number of graphic formats for you, the results are often better when you use a third-party graphic tool such as Graphic Workshop. See §5.7 [Processing graphics](#) on page 126 for more information.

If you are converting to Word, be aware of version-dependent image scaling issues; see §6.14.7 [Preserving graphics scale in Word](#) on page 191.

2.5.3 Replacing embedded graphics with referenced graphics

Mif2Go can export embedded graphics from your FrameMaker document.

*Referenced
graphics are
preferable*

If your document contains many images that were imported into FrameMaker by copying instead of by reference, you might want to consider using **Mif2Go** to export them as graphics files. Then you can re import them as referenced graphics. Referenced graphics do not have to be processed again and again as you fine-tune your **Mif2Go** conversion settings. Importing graphics into your FrameMaker documents by reference is a good idea anyway, if your document design and workflow permit doing so.

Note: If the embedded images are embellished in FrameMaker with callouts or other added features, you will have to replace those features for the reimported graphics.

*Make sure file
names will be
unique*

Because FrameMaker does not retain the original names of embedded images, each exported file comes out named with the first four letters of the FrameMaker file name followed by an incremental four-digit number; for example, `intr0001.gif`. To keep all the file names distinct you can change how many letters and digits **Mif2Go** uses for a name; see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134.

*Export and
replace*

To replace embedded graphics with referenced graphics:

1. Set up **Mif2Go** for Standard HTML (or any other HTML/XML format); see §3.3 [Creating a Mif2Go conversion project](#) on page 78.

2. In your starting project configuration file, set the following options:

```
[GraphExport]
ImportGraphics=Export
ExportNameChars=n
ExportNumDigits=m
```

where *n* is the number of letters to use in the name of an exported file and *m* is the number of digits. The default value is 4 for each of these options. See §4.4 [Understanding the rules for configuration settings](#) on page 102.

3. Run the conversion with default export settings; see §3.6 [Converting documents](#) on page 82.
4. (Optional) Change the name of each newly exported graphics file to a more reasonable name that helps you recognize the graphic.
5. (Optional) Move the newly exported graphics files to a directory relative to the directory that contains your FrameMaker document.
6. For each embedded graphic in each FrameMaker file in your document:
 - 6.1. Select the image itself (*not* the frame it is in).
 - 6.2. Use **File > Import > File** to replace the embedded image with the corresponding graphics file, imported by reference.

For additional information see §31.2.3.2 [Replacing embedded graphics](#) on page 878.

2.5.4 Setting up cross references to and from text insets

The FrameMaker process for updating text insets is like the process for generating contents, index, and other lists: any properties you apply to the container document *for* the inset (rather than *to* the inset) are cleared away whenever you update the document. To create persistent cross references, the container document must reference markers instead of paragraphs.

Also, if a text inset references the container document (or another text inset), and you import the text inset into more than one container document, you must create a separate cross-reference pointer in the text inset for each such container document.

Note: Text-inset referencing is a FrameMaker issue, not a **Mif2Go** issue. Improperly created text-inset cross references fail in FrameMaker itself, and in any output type, as soon as you update the container document.

In this section:

§2.5.4.1 [Creating persistent references to text insets](#) on page 71

§2.5.4.2 [Creating persistent references from text insets](#) on page 71

§2.5.4.3 [Creating persistent references between text insets](#) on page 72

2.5.4.1 Creating persistent references to text insets

To cross-reference material in a text inset from a container document so that the cross-reference links survive document updating:

1. Open both the container document and the text-inset file in FrameMaker.
2. From the container document, import the text inset (by reference) at the point where you want the inset to appear.
3. From the container document, use the FrameMaker *Cross-Reference* dialog to create a cross reference to the appropriate paragraph in the text inset:
 - 3.1. From the **Document** list, choose the name of the text-inset file (*not Current*).
 - 3.2. From the **Source Type** list, choose **Paragraph**.
 - 3.3. Select the text-inset format and paragraph you want to reference.
 - 3.4. Select the appropriate cross-reference format.
 - 3.5. Click **Insert**; FrameMaker inserts a cross-reference marker *into the text-inset file*, in the paragraph you selected.
4. Save the text-inset file.
5. In the container document, use the FrameMaker *Cross-Reference* dialog to replace the cross reference you created in [Step 3](#):
 - 5.1. From the **Document** list, choose **Current**.
 - 5.2. From the **Source Type** list, choose **Cross-Reference Marker**. (The marker list shows all the markers in text insets, along with any in the container document.)
 - 5.3. Select the marker for the cross reference you created in [Step 3](#).
 - 5.4. Click **Insert**; FrameMaker inserts *into the container file* a cross-reference to the text-inset marker, but with the paragraph properties of the container document.

The resulting cross reference will persist through document updates.

2.5.4.2 Creating persistent references from text insets

To cross-reference material in container documents from a text inset so that cross-reference links survive document updating regardless of which container document imports the inset:

1. Open the text-inset file in FrameMaker.
2. Open a container document.

3. From the text-inset file, use the FrameMaker *Cross-Reference* dialog to create a cross reference to the appropriate paragraph in the container document:
 - 3.1. From the **Document** list, choose the name of the container file (*not Current*).
 - 3.2. From the **Source Type** list, choose **Paragraph**.
 - 3.3. Select the container-document format and paragraph you want to reference.
 - 3.4. Select the appropriate cross-reference format.
 - 3.5. Click **Insert**; FrameMaker inserts a cross-reference marker *into the container file*, in the paragraph you selected.
4. Save the container file.
5. In the text-inset file, use the FrameMaker *Cross-Reference* dialog to replace the cross reference you created in [Step 3](#):
 - 5.1. From the **Document** list, choose the name of the container file (*not Current*).
 - 5.2. From the **Source Type** list, choose **Cross-Reference Marker**.
 - 5.3. Choose the marker for the cross reference you created in [Step 3](#).
 - 5.4. Click **Insert**; FrameMaker inserts *into the text-inset file* a cross-reference to the marker in the container document.
6. In the text-inset file, use the FrameMaker *Conditional Text* dialog to define a unique condition, and apply that condition to the cross-reference pointer.
7. Repeat [Step 2](#) through [Step 6](#) for each container document that imports the text inset, defining a *different* condition for each, and applying that condition to the newly created cross-reference pointer in the text inset.
8. In *each* container document, do the following:
 - 8.1. Define *all* the unique text-inset conditions you created in [Step 6](#).
 - 8.2. **Show** the condition for the container document in question, and **Hide** all the other unique text-inset conditions.

2.5.4.3 Creating persistent references between text insets

To cross-reference material in one text inset from another text inset so that the cross-reference links survive document updating regardless of which container document imports the insets:

1. Open both text insets in FrameMaker.
2. In the referenced text inset, create a cross-reference target marker (see §2.5.4.1 [Creating persistent references to text insets](#) on page 71).
3. In the referencing text inset, place a distinct cross-reference pointer for each different container document, each pointing to the marker you created in [Step 2](#).
4. Use conditional text to define and apply a distinct condition to each such cross-reference pointer in the referencing text inset; see §2.5.4.2 [Creating persistent references from text insets](#) on page 71.
5. In *each* container document, do the following:
 - 5.1. Define *all* the unique text-inset conditions you created in [Step 4](#).
 - 5.2. **Show** the condition for the container document in question, and **Hide** all the other unique text-inset conditions.

2.5.5 Creating hotspots for hypertext links

To create a hypertext hotspot:

- Insert an appropriate hypertext marker in your FrameMaker document where you want the link to start. See §34.1.2 [Using markers to add links and instructions](#) on page 935.
- Apply a character format to the surrounding text to delimit the hotspot for the link.

To make an entire paragraph into a hotspot, do not apply *any* character formats to the paragraph. This is intended for short paragraphs (TOC items, headings, and so forth), where adding a character format would be an annoyance.

See §5.10 [Creating hotspots for hypertext links](#) on page 138.

2.5.6 Producing a single output file from a FrameMaker book

When you convert a FrameMaker book, **Mif2Go** does not create an output file from the FrameMaker book file itself; nor does **Mif2Go** combine all the FrameMaker files in a book into a single output file. Instead, each file in a FrameMaker book is converted to one or more output files. Each FrameMaker chapter contains its own formatting information, which can be entirely different from that in other chapters. For this reason, chapters must be converted to separate output files.

To produce one file from an entire book with **Mif2Go**, you must first combine the chapter files in FrameMaker. This can require reworking links and numbering. You might also have to deal with issues such as differing definitions of formats and variables in different chapters. You have these options:

[Use a script to preserve links and numbering](#)

[Import chapters as text insets \(and lose links\).](#)

*Use a script to
preserve links
and numbering*

The process of combining book components into a single document can be automated with FrameScript, ExtendScript (FrameMaker 10 or a later version), or the FDK. A script can modify cross references, hypertext links, and numbering sequences so they are internal to the combined file, and resolve differences in variable definitions. FrameScript is available from Finite Matters Ltd:

<http://www.framescript.com/>

*Import chapters
as text insets
(and lose links)*

Otherwise, to use **Mif2Go** to produce a single output file from a FrameMaker book:

1. In FrameMaker, create a new file from the chapter template for your book.
2. Import all chapter files into the new file by reference, as text insets.

Whenever you want to produce a combined file, convert the combined FrameMaker file. If you change any content in the original FrameMaker book, the only maintenance required for the combined file is to update insets. This method allows you to:

- select only files you want to include
- add other information as needed
- combine the files only once.

You will not have working cross references, of course, because they reference the individual chapters and not the container. And the numbering, if any, is unlikely to work.

2.5.7 Preparing a structured document for conversion

You can use **Mif2Go** to convert Structured FrameMaker files. **Mif2Go** fully supports structured cross references; however, **Mif2Go** does not use structure tags and attributes for any other purpose. You must use distinct format names to get distinct effects in the output. We consider it better practice to separate structure and formatting by having a set of

formats rather than a mass of overrides. Using distinct formats simplifies maintenance by letting a designer modify templates without touching the structured application.

At present, elements and attributes created as part of the structure are not included in XML output. Starting with FrameMaker version 7.1, you can export XML directly from FrameMaker.

See also:

§5.8 [Converting structured documents](#) on page 135

§14.1 [Understanding how Mif2Go generates XML output](#) on page 457.

2.6 Establishing a conversion environment

If your FrameMaker files are on a network server, you might find that FrameMaker sometimes has trouble accessing those files. It is best to make a local copy, and use **Mif2Go** to convert from that copy.

Follow these steps to establish a working environment for your conversion project:

1. **Establish a conversion project directory.** Unless you specify a directory, **Mif2Go** places output files in the same directory as the FrameMaker files to be converted. *We strongly advise you to create a separate project directory for each conversion project*, under the directory where your FrameMaker files are located (see §3.3 [Creating a Mif2Go conversion project](#) on page 78). If you plan to convert the same FrameMaker document to several different output types, create a separate project directory for each, to avoid overwriting files.

2. **Decide about a configuration file:**

New project: If you are setting up a brand-new conversion project, remove from the project directory any existing configuration file intended for the same output type.

Existing project: If you want **Mif2Go** to use a configuration file already created for the document, or one based on a different FrameMaker document, place that configuration file (or a copy of it) in the project directory.

*Restart
FrameMaker
between
conversions*

3. **Restart FrameMaker.** If you have already set up a **Mif2Go** conversion in your current FrameMaker session, *close FrameMaker and restart it*. Sometimes **Mif2Go** “remembers” settings used in a prior conversion, causing unexpected results (such as crashes). Whether this is likely to happen seems to depend on your operating environment and on **Mif2Go** configuration settings. In some situations you might be able to get away without restarting FrameMaker, especially if you delete output files and conversion files (but *not* project or configuration files) between runs; see §C.2.4 [What not to do with conversion files](#) on page 1025. You should never have to reboot the system, unless you have hit the Windows 9x/ME GDI resource-leak wall; see §8.6.2 [Avoiding the GDI resource leak](#) on page 264.
4. **Close previous output files.** If you are converting to Word, close any RTF files from a previous conversion that you have open in Word. **Mif2Go** cannot write a new version if Word has the old one open.
5. **Select a book or document.** In FrameMaker, open and select a book or document file.
6. **Start Mif2Go.** Follow instructions in §3 [Converting a book or document](#) on page 77.

2.7 Setting up multiple interlinked HTML projects

If you wish to convert two or more FrameMaker books to the same HTML output type, and these books contain cross references or hypertext links to each other, you have two basic choices for a conversion environment:

- Use the same project directory for all books
- Use a different project directory for each book.

The first is the simpler choice: all output files from all the books go into the same project directory. Use a single project configuration file for all the projects, with each of the book project files pointing to the project directory. This way **Mif2Go** will automatically handle all the links between books.

However, if some books include files that have the same name but different content (for example, `intro.fm`), the files with duplicate names will get overwritten by whichever book is converted last; for the other books, the content will not be correct. In that case, you will need to use a separate project directory for each book, so each has its own configuration file. Then you must tell each book where the files in the other books are located; see §19.6 [Linking to other files and other Mif2Go projects](#) on page 621.

2.8 Preparing deliverables after conversion

Converting a document might be just one step in a workflow that also includes preparing the results for distribution. **Mif2Go** can automate much of this post-conversion processing. For many conversions, **Mif2Go** supports one-click production of deliverables; see §35 [Producing deliverable results](#) on page 955. For more involved workflows, **Mif2Go** supports interactive prompts and custom before-and-after processing; see §34 [Automating Mif2Go conversions](#) on page 933.

3 Converting a book or document

This section shows how to use the **Mif2Go** FrameMaker plug-in to set up a conversion project and convert FrameMaker books and documents. Topics include:

- §3.1 [Checking set-up and conversion requirements](#) on page 77
- §3.2 [Starting Mif2Go](#) on page 77
- §3.3 [Creating a Mif2Go conversion project](#) on page 78
- §3.4 [Choosing project set-up options](#) on page 79
- §3.5 [Understanding how Mif2Go sets up a project](#) on page 82
- §3.6 [Converting documents](#) on page 82
- §3.7 [Choosing final conversion options](#) on page 83

See also:

- §2.5 [Preparing documents for conversion](#) on page 69
- §2.6 [Establishing a conversion environment](#) on page 74

3.1 Checking set-up and conversion requirements

Before you set up a **Mif2Go** conversion project, check the following requirements:

[Mif2Go installed](#)

[No spaces in file names or paths](#)

[No configuration file before project set-up](#)

Mif2Go installed

Mif2Go must be installed on your system as a FrameMaker plug-in; see §1.3.3 [Install Mif2Go](#) on page 56.

No spaces in file names or paths

All FrameMaker files, graphics files, and any other files to be used in the conversion should have paths and names that do *not* contain:

- spaces
- multiple periods.

See §2.1 [Naming files, directories, and paths](#) on page 65.

No configuration file before project set-up

If you are creating a *new* project, in order to use a **Mif2Go Set Up** dialog there must be *no prior configuration file for the same output type* present in the project directory; see §3.4 [Choosing project set-up options](#) on page 79.

3.2 Starting Mif2Go

To convert a FrameMaker book, run **Mif2Go** with the book file active. To convert a single-file document, run **Mif2Go** with the document file active.

1. Select the book or document file in FrameMaker.
2. From the FrameMaker File menu, choose one of the following:
 - Set Up Mif2Go Export...** to convert a book or document for the first time.
 - Save Using Mif2Go...** for subsequent conversions of the same project.

Individual chapters

To reconvert a chapter of a book (or to convert a newly added chapter), after you have set up the book conversion *keep the book file open*, and run **Save Using Mif2Go...** from the chapter file.

3.3 Creating a Mif2Go conversion project

Mif2Go stores project information in a file that has the same base name as your FrameMaker document, with extension `.prj`. This project file is located in the same directory as your FrameMaker document.

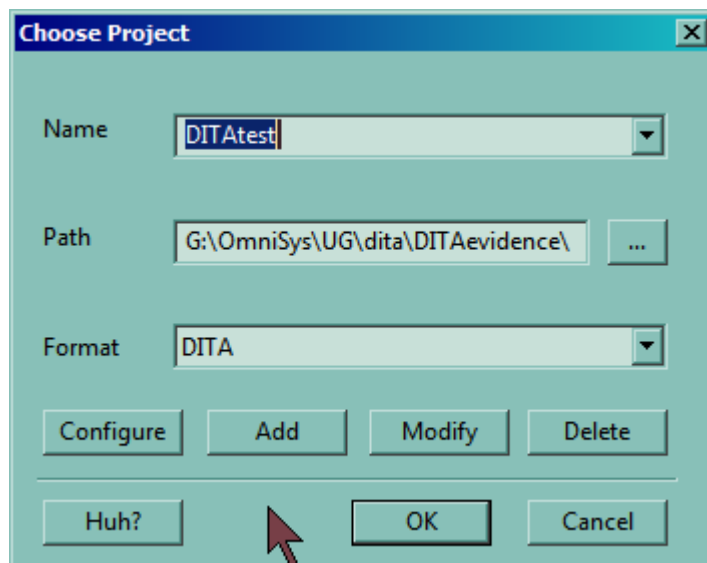
To specify a conversion project, from the document or book file click one of the two **Mif2Go** options on the FrameMaker **File** menu:

Set Up Mif2Go Export... to create a new project

Save Using Mif2Go... to modify or re-run an existing project.

The **Mif2Go Choose Project** dialog shown in [Figure 3-1](#) opens.

Figure 3-1 Choose Project dialog



<u>N</u>ame	Give your project a unique name , or select an existing project from the list. The default name for a new project is the base name of your FrameMaker book or document file. Your project list is kept in a file with extension <code>.prj</code> , located in the same directory as your FrameMaker files.
<u>P</u>ath	Select or create a project directory for the converted files. Browse (...) for the directory where you want Mif2Go to place output files, or create a new directory. For best results, specify a directory immediately under the directory where your FrameMaker document is located. If you plan to convert the same document to several different output types, create a separate project directory for each.
<u>F</u>ormat	Select an output type from the list.
<u>C</u>onfigure	Start the Mif2Go Configuration Manager to access the configuration files your project references. See §4.2 Editing files with the Configuration Manager on page 91.
<u>A</u>dd	To add a new project, click Add . When you click OK , the project is added to your project list.
<u>M</u>odify	To change the name, directory, or format of an existing project, click Modify . When you click OK , information for the currently displayed project is changed in your project list.

Delete	To delete the currently displayed project, click Delete . When you click OK , all information for the project is removed from your project list.
Huh?	To see this information in HTML Help while you are working, with Mif2Go file %OMSYSHOME%\m2g\usersguide\ugmif2go.chm open (see §1.3.3 Install Mif2Go on page 56), click Huh?
OK	To save changes you made to the currently displayed project, Click OK . The <i>Choose Project</i> dialog is dismissed.
Cancel	To discard current changes, click Cancel ; you lose any additions and changes you just made. You might want to do this if you accidentally delete the wrong project, and realize it before you click OK .
<i>Proceed with the conversion</i>	What happens after you dismiss the <i>Choose Project</i> dialog depends on whether you are creating a new conversion project or modifying an existing project:
<i>New project:</i>	If this is a <i>new</i> project (no project configuration file is present in the project directory), you get a one-time opportunity to specify set-up options. See §3.4 Choosing project set-up options on page 79.
<i>Existing project:</i>	If this is an existing project (a project configuration file is already present in the project directory), you must use a text editor to modify conversion settings.

3.4 Choosing project set-up options

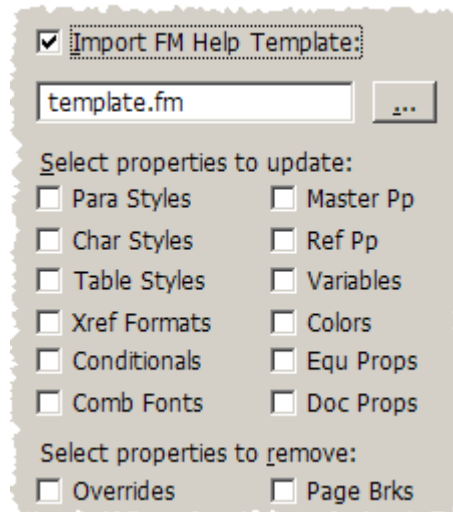
Mif2Go presents a different *Set Up* dialog for each output type.

In this section:

- §3.4.1 [Importing formats from a FrameMaker template](#) on page 79
- §3.4.2 [Converting FrameMaker system variables to text](#) on page 80
- §3.4.3 [Generating and updating your document](#) on page 81
- §3.4.4 [Including FrameMaker-generated files](#) on page 81
- §3.4.5 [Understanding configuration settings for general set-up options](#) on page 81
- §3.4.6 [Choosing output-specific set-up options](#) on page 82

3.4.1 Importing formats from a FrameMaker template

To import formats from a FrameMaker conversion template, check **Import FrameMaker Template** (or **Import FM Help Template**) in the *Set Up* dialog as shown in [Figure 3-2](#), and browse to the location of the template file.

Figure 3-2 Import FrameMaker Template

Mif2Go Set Up dialogs provide checkboxes for all the same properties as the FrameMaker *Import Formats* dialog. Some cautions:

- If you specify a template file, but you do not check any properties, *all properties except Document properties* are imported, and overrides and page breaks are removed.
- Best not to check **Doc Props** (document properties); a defect in FrameMaker can cause unexpected changes to document settings.
- The **Comb fonts** option stands for “combined Japanese/English fonts”. Although **Mif2Go** supports Japanese on FrameMaker version 8 and later versions, checking this option can cause “template failure” on non-Japanese systems. If this happens, **Mif2Go** tries the template import again, without the **Comb fonts** option.

See also:

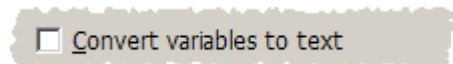
[§2.4 Importing formats from a conversion template](#) on page 67

[§3.4.5 Understanding configuration settings for general set-up options](#) on page 81

[§30.7 Applying FrameMaker conversion templates](#) on page 863

3.4.2 Converting FrameMaker system variables to text

To convert the FrameMaker date, time, and file-name system variables in your document to text before conversion, check **Convert variables to text** in the *Set Up* dialog, as shown in [Figure 3-3](#).

Figure 3-3 Convert variables to text

Only system date/time and file-name variables *on body pages* are converted. Other variables are already present in a usable form in MIF files. Use this option conservatively, because it can drastically increase the time required to convert your document.

See also:

[§3.4.5 Understanding configuration settings for general set-up options](#) on page 81

[§5.1.9 Converting system variables to text](#) on page 114

3.4.3 Generating and updating your document

To generate and update your document after importing formats (if you checked that box) and before carrying out the conversion, check **Generate/Update after import** in the *Set Up* dialog. Use this option only if necessary; mid-conversion updating takes time and requires all files to be open at once.

See also:

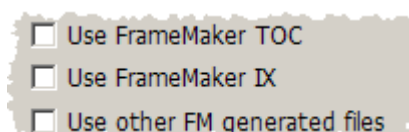
§3.4.5 [Understanding configuration settings for general set-up options](#) on page 81

§5.6 [Generating/updating before converting](#) on page 126

3.4.4 Including FrameMaker-generated files

To include FrameMaker-generated files in the conversion, check one or more of the *Set Up* dialog boxes shown in [Figure 3-4](#).

Figure 3-4 Include generated files



You do not need the TOC or IX files if you are creating WinHelp or HTML-based help, because **Mif2Go** generates contents and index for Help systems.

See also:

§3.4.5 [Understanding configuration settings for general set-up options](#) on page 81

§5.5 [Converting FrameMaker-generated files](#) on page 124

§7.3.2 [Including FrameMaker TOC and IX in Help systems](#) on page 205

3.4.5 Understanding configuration settings for general set-up options

[Table 3-1](#) shows which configuration settings correspond to each of the general set-up options described in sections [3.4.1](#) through [3.4.4](#), and shows the default value for each setting.

Table 3-1 General set-up options and settings

Set-up dialog Option	Configuration file [Setup] Setting	Default	Ref.
Import FrameMaker Template	ApplyTemplateFile=Yes	No	30.7.1
	TemplateFileName=pathtofile.fm	None	30.7.1
Select properties to update, and properties to remove	AppliedTemplateFlags= <i>a number you must compute; default is 0 (zero): all but Document properties</i>	All but Doc props	30.7.1
Convert system variables to text	ConvertVariables	No	5.1.9
Generate/Update after import	GenerateBook	No	5.6
Use FrameMaker TOC	UseFrameTOC (Print RTF, HTML, XML)	Yes	5.5.1
	UseFrameTOC (WinHelp, HTML-based help, DITA)	No	7.3.2
Use FrameMaker IX	UseFrameIX (Print RTF, HTML, XML)	Yes	5.5.1
	UseFrameIX (WinHelp, HTML-based help, DITA)	No	7.3.2
Use other FM generated files	UseFrameGenFiles (all but DITA)	Yes	5.5.2

3.4.6 Choosing output-specific set-up options

For set-up options specific to each output type, see:

<u>Output type</u>	<u>Link to set-up options</u>
ASCII DCL	§38.3.1 Setting up an ASCII DCL project on page 1009
DITA XML	§15.2.2 Choosing set-up options for a DITA XML project on page 479
DocBook	§17.2.2 Choosing set-up options for a DocBook project on page 560
Eclipse Help	§12.2.2 Choosing set-up options for an Eclipse Help project on page 404
HTML Help	§9.3.2 Choosing set-up options for an MS HTML Help project on page 298
HTML/XHTML/XML	§13.2.2 Choosing set-up options for an HTML or XHTML project on page 425
JavaHelp/Oracle Help	§11.3.2 Choosing set-up options for a JavaHelp or Oracle Help project on page 375
MIF	§38.2.2 Setting up a FrameMaker MIF project on page 1006
OmniHelp	§10.3.2 Choosing set-up options for an OmniHelp project on page 346
WinHelp	§8.2.2 Choosing set-up options for a WinHelp project on page 244
Word/WordPerfect	§6.2.2 Choosing set-up options for a print RTF project on page 146

3.5 Understanding how Mif2Go sets up a project

When you click **OK** on a *Set Up* dialog, **Mif2Go** does the following:

1. Copies from %OMSYSHOME%\m2g\local\starts a starting project configuration file for the output type you specified, and places this file in the project directory; see §4.3 [Understanding where project settings come from](#) on page 102.

Note: If a configuration file for the same output type is already present in the project directory when you start **Mif2Go**, you will not see a *Set Up* dialog.

2. If not already present, creates a document-specific configuration file for your FrameMaker document; see §30.3 [Including document-specific configuration files](#) on page 852.
3. Includes in the starting project configuration file (see [Step 1](#)) a reference to the document-specific configuration file.
4. Loads the new project configuration file in Notepad for you to inspect and edit.

Once you click **OK** on the *Set Up* dialog, thereafter *you must use a text editor to change any settings in these or any other configuration files.*

3.6 Converting documents

After setting up a conversion project, follow these steps to convert a FrameMaker document:

1. **Open a document in FrameMaker.** To convert all the files in a book, select the book file. Even if a few files in the book do not need converting, this is usually faster than converting each file separately.
2. **Select a conversion project.** On the FrameMaker **File** menu, choose **Save Using Mif2Go...** The *Choose Project* dialog opens (see [Figure 3-1](#) on page 78). If the project in the **Name** field is not the one you want, select your project from the list. Click **OK** to dismiss the dialog.

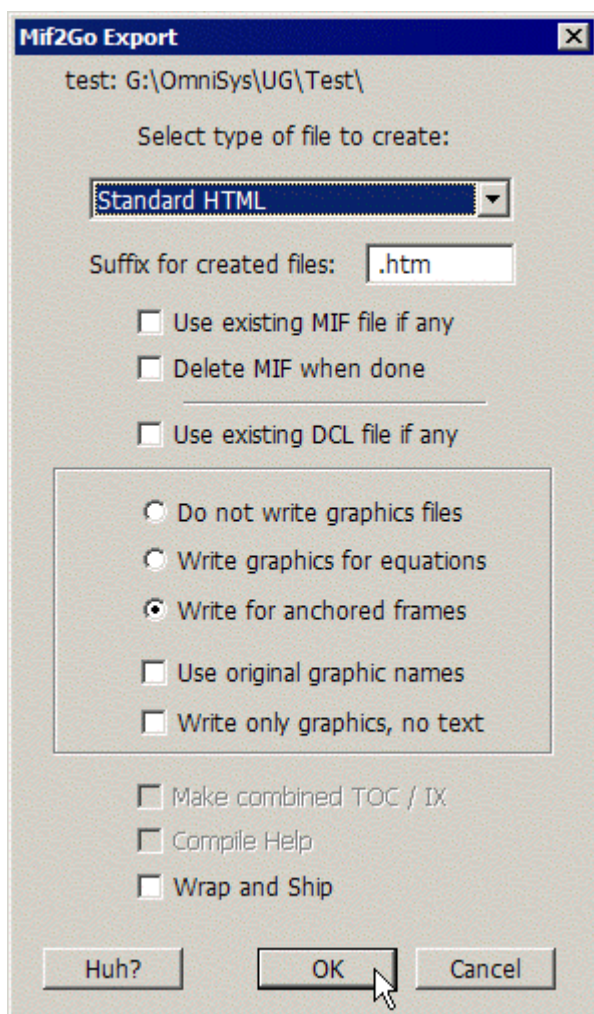
3. **Choose export options.** When you dismiss the *Choose Project* dialog, the **Mif2Go Export** dialog opens (see [Figure 3-5](#) on page 83). For information about export options, see §3.7 [Choosing final conversion options](#) on page 83.
4. **Start the conversion.** Click **OK** in the **Mif2Go Export** dialog to start converting documents.

During conversion, **Mif2Go** displays progress messages in the status bar at the bottom of the book or file window.

Run the conversion the first time with just the settings you specified in the *Set Up* dialog, then fine-tune after you examine the output. Always make the minimum number of changes, and only after trying out the default values; Omni Systems developers work hard to make the defaults reasonable for most purposes.

You can keep a text editor open while trying different settings, using **File > Save** in the editor to update the configuration file (see §2.3 [Understanding Mif2Go configuration files](#) on page 66), then **File > Save Using Mif2Go...** in FrameMaker to run the conversion again.

Figure 3-5 Mif2Go Export dialog



3.7 Choosing final conversion options

Use the *Export* dialog ([Figure 3-5](#)) to make last-minute changes to conversion options.

In this section:

- §3.7.1 [Understanding how export options work](#) on page 84
- §3.7.2 [Specifying output type and file extension](#) on page 84
- §3.7.3 [Choosing input source and disposition](#) on page 85
- §3.7.4 [Figuring out graphics export options](#) on page 85
- §3.7.5 [Choosing postprocessing options](#) on page 88

3.7.1 Understanding how export options work

Export options both reflect and affect configuration settings:

- When you change an option in the *Export* dialog (see [Figure 3-5](#)), **Mif2Go** makes the corresponding change to relevant settings in your project configuration file.
- When you change a relevant setting in your project configuration file, next time you open the *Export* dialog it will show this change to the corresponding option.

When you rerun a conversion, the *Export* dialog shows the options you checked the last time you ran the conversion; unless, in the meantime, you altered the corresponding settings in the project configuration file. [Table 3-2](#) lists the configuration-file settings that correspond to each export option. Default values that differ from the corresponding set-up value are shown in red.

Table 3-2 Mif2Go export options and configuration settings

Export dialog Option	Configuration file		Default	References
	Section	Setting		
Select type of file to create	(None)	(See Figure 3-1 on page 78 for a list)	(None)	3.3
Suffix for created files	[Setup]	FileSuffix = .any	(Varies)	5.1.1
Use existing MIF file if any	[Setup]	UseExistingMIF = Yes	No	5.1.3
Use existing DCL file if any	[Setup]	UseExistingDCL = Yes	No	5.1.4
Do not write graphics files	[Graphics]	UseGraphicPreviews = No	Yes	5.7.2
	[Setup]	WriteEquations = No	No	
	[Setup]	WriteAllGraphics = No	No	
Write graphics for equations	[Graphics]	UseGraphicPreviews = No	Yes	5.7.2.1, 31.2.5.3
	[Setup]	WriteEquations = Yes	No	
	[Setup]	WriteAllGraphics = No	No	
Write for anchored frames	[Graphics]	UseGraphicPreviews = Yes	Yes	5.7.2.2, 31.2.5.3
	[Setup]	WriteEquations = No	No	
	[Setup]	WriteAllGraphics = Yes	No	
Use original graphic names	[Graphics]	UseOriginalGraphicNames = Yes	No	23.4.1, 31.2.5.4, 31.3.1.4, 31.3.1.5
Write only graphics, no text		No corresponding setting		none
Make combined TOC/IX	[HelpOptions]	MakeCombinedCnt = Yes	Yes	8.2.8
Compile Help	[Automation]	CompileHelp = Yes	No	8.2.8, 9.14.1
Wrap and Ship	[Automation]	WrapAndShip = Yes	No	35.2

3.7.2 Specifying output type and file extension

Output type To override the output type you originally specified for a project, on the **Mif2Go** *Export* dialog select a different format from the list, under:

Select type of file to create

You must select a format that is valid for the configuration file associated with the project. [Table 3-2](#) on page 84 shows the settings that are valid for each type of configuration file.

File extension When the *Export* dialog opens, this entry:

Suffix for created files

shows the file extension specified in the configuration file. However, if you change the output type, the file extension automatically changes to the appropriate value for the new output type you select. In either case, you can type in a different file extension.

MIF or DCL If you specify `.mif` as the file extension, **Mif2Go** stops after creating MIF files; and if you specify `.dcl`, **Mif2Go** stops after creating DCL files.

If you specify `.mif` and you are converting a FrameMaker book; and the names of files in the book have extensions that start with `.fm` (such as `.fm`, `.fm5`, or `.fm6`); **Mif2Go** uses the original book file extension (typically `.book` or `.bk`) for the MIF book file.

3.7.3 Choosing input source and disposition

Existing MIF files If you already have *current* MIF files for your document located in the project directory, you can save conversion time by checking:

Use existing MIF file if any

in the **Mif2Go** *Export* dialog.

This setting is meant to save you time when you are experimenting with different configuration-file settings for the same output. When this box is checked, changes made to FrameMaker binary files *after* the MIF files were created are not included in the conversion. For production, check this box only if you are sure of the following:

- the FrameMaker file(s) in question have not been altered
- you have not changed an imported template, or template import option.

If this box is checked but **Mif2Go** cannot find one or more MIF files in the project directory, **Mif2Go** creates from the FrameMaker binary files any that are missing.

If you have only MIF files, but not the original FrameMaker binary files, *always* check this box.

Existing DCL files If you already ran the conversion with **ASCII DCL only** as the output type, you can check:

Use existing DCL file if any

to avoid redoing the first part of the conversion, and also avoid rewriting graphics.

Delete MIF files? If you are short on disk space, check:

Delete MIF when done

Mif2Go creates and removes MIF files one at a time. However, if you have *only* MIF files, and you do not have the original FrameMaker binary files, *do not* check this option.

3.7.4 Figuring out graphics export options

Your conversion must use one of the following three graphics options presented in the **Mif2Go** *Export* dialog:

- | | |
|-------------------------------------|--|
| Do not write graphics files | (because they already exist) |
| Write graphics for equations | (use Mif2Go native graphics processing) |
| Write for anchored frames | (use FrameMaker export filters for graphics) |

Two additional graphics options are also available in the **Mif2Go** *Export* dialog:

- | | |
|-----------------------------------|--|
| Use original graphic names | (HTML only: avoid FrameMaker export filters) |
|-----------------------------------|--|

Write only graphics, no text (to experiment with different graphics settings)

Table 3-2 on page 84 shows the configuration-file settings that correspond to each of these options.

In this section:

§3.7.4.1 [Omitting and restoring graphics production](#) on page 86

§3.7.4.2 [Using Mif2Go native graphics processing](#) on page 86

§3.7.4.3 [Using FrameMaker export filters](#) on page 87

§3.7.4.4 [Using original referenced graphics for HTML](#) on page 87

§3.7.4.5 [Producing only graphics](#) on page 88

3.7.4.1 Omitting and restoring graphics production

Omit graphics processing

If you have already converted (or provided replacements for) the graphics in your document, you might save processing time by choosing **Do not write graphics files** in the **Mif2Go Export** dialog. However, this choice *has no effect* if your configuration file specifies **Mif2Go** native graphics processing, such as for WinHelp output. Only production of graphics using FrameMaker export filters is omitted.

Two effects are involved:

- the actual production of graphics files
- the way those files are referenced in the output.

If you choose **Do not write graphics files** the *first* time you run a conversion, **Mif2Go** creates references based on file names that would have been assigned to graphics files produced using FrameMaker export filters, but without actually producing any graphics.

When you choose **Do not write graphics files** for a *subsequent* conversion run, **Mif2Go** checks the settings in the configuration file, and creates references based on the more recently used of the two graphics production methods.

Restore graphics processing

If you add or change any graphics in your document after you have run the conversion with **Do not write graphics files** checked, and you have not specified external replacement files for the graphics (see §31.3 [Replacing and relocating graphics files](#) on page 887), you must choose one of the other two options.

Choose the correct option

If you do not remember which option you chose when you first ran the conversion, look in the [Graphics] section of the configuration file for the UseGraphicPreviews setting:

<u>[Graphics] setting</u>	<u>Export option</u>	<u>Production method</u>
UseGraphicPreviews=Yes	Write for anchored frames	FrameMaker export filters
UseGraphicPreviews=No	Write graphics for equations	Mif2Go native graphics processing

3.7.4.2 Using Mif2Go native graphics processing

Choose **Write graphics for equations** if you do *not* want **Mif2Go** to use FrameMaker export filters to convert graphics. This is an appropriate choice if either of the following is true:

- You are relying on **Mif2Go** native graphics processing to produce WMF graphics for RTF output (see §5.7.2.1 [Using Mif2Go native graphics processing](#) on page 128).
- You have specified export options for *embedded* graphics (see §31.2.3 [Exporting and converting embedded graphics](#) on page 877), and your *referenced* graphics are either:
 - already in a suitable format (see §31.1 [Choosing an appropriate graphics format](#) on page 869), or

- mapped to suitable equivalents (see §31.3 [Replacing and relocating graphics files](#) on page 887).

However, if you are converting to HTML, the following graphics in your document *are not converted* when you choose **Write graphics for equations**:

- illustrations created with FrameMaker drawing tools
- graphics in reference-page frames
- compound graphics (referenced or embedded) that include callouts, text lines, or FrameMaker native graphic elements.

Note: **Mif2Go** uses FrameMaker export filters to convert FrameMaker native math equations, regardless of which graphics-processing option you choose.

3.7.4.3 Using FrameMaker export filters

When you choose **Write for anchored frames**, **Mif2Go** uses the FrameMaker export filters to create an external file for every anchored frame in your document, including:

- all graphics, whether embedded or imported by reference
- any tables in anchored frames
- all equations.

Choose this option if either of the following is true:

- You are converting to HTML or XML, and any of your graphics include FrameMaker drawing elements such as callouts.
- Your graphics are not already in a suitable format (see §31.1 [Choosing an appropriate graphics format](#) on page 869), and you do not have a graphics conversion program available.

For more information about when to have **Mif2Go** use the FrameMaker export filters, see §5.7.2 [Choosing how to convert graphics](#) on page 127.

3.7.4.4 Using original referenced graphics for HTML

For referenced graphics, checking **Use original graphic names** in the **Mif2Go Export** dialog directs **Mif2Go** to use, wherever possible, the original external graphics file instead of any file created from the graphic by FrameMaker export filters; that is, for any graphic that is alone in its anchored frame, without FrameMaker-added elements. The resulting `` tag does the following:

- References the original file name of the graphic (possibly modified for path and extension by other configuration settings; see §31.3 [Replacing and relocating graphics files](#) on page 887).
- Uses the FrameMaker-determined size of the original referenced image as opposed to the size of the enclosing anchored frame (to preserve scale, possibly modified by other configuration settings; see §23.9 [Scaling images for HTML](#) on page 719).

You might want to check this option if your conversion project has the following characteristics:

- Output type is HTML (HTML, XHTML, XML, or HTML-based help).
- At least some of the graphics in your document are all of the following:
 - either:
 - › referenced (not embedded in the document), or:
 - › already exported to external files (see §31.2.3 [Exporting and converting embedded graphics](#) on page 877)
 - either:

- › in a suitable format (such as JPEG, GIF, or PNG), or:
- › replaced by alternate files you specified via configuration-file settings, where the replacements have the same base file names as the originals
- alone in their anchored frames (no callouts, text lines, or other images).

If you choose **Write for anchored frames** and check **Use original graphic names** in the **Mif2Go Export** dialog, you get the best of both worlds: **Mif2Go** uses original (or replaced) external graphics wherever possible, and also uses FrameMaker export filters to convert graphics for which there is no external name (FrameMaker native graphics, embedded graphics, and reference-page graphics).

Note: If an image is not the same size as the enclosing anchored frames, the size used when you check **Use original graphic names** is that of the image, not of the frame, to avoid distorting the image.

See also:

§23.4.1 [Using referenced graphics without converting](#) on page 706

§23.4.4 [Using referenced, embedded, and compound graphics](#) on page 707

§31.3.1 [Changing graphics files for HTML output](#) on page 887

3.7.4.5 Producing only graphics

After you have successfully converted your document, to experiment with different graphics DPI or format settings without actually producing document files, check **Write only graphics, no text** in the **Mif2Go Export** dialog. When you check this option, **Mif2Go** uses the .grx files from the previous conversion run to get a list of graphics to produce. You must have already run the conversion at least once to produce the .grx files, because this option works only when .grx files are present in the project directory. See §C.2.2 [Files created during conversion](#) on page 1021.

If you really want to output only graphics, without first producing .grx files, you could set [Setup]GraphicsFirst=Yes in the configuration file; then instead of referring to .grx files, **Mif2Go** would crank out every graphic in your document, including unused reference-page and master-page items (but not any graphics in hidden conditional text). See §5.7.3.1 [Processing all graphics first](#) on page 132.

This process is really useful only for RTF output; for HTML and XML, setting [Setup]GraphicsFirst=Yes is usually a waste of time and space. And if you decide to change graphics formats, you must run the entire conversion again (possibly with **Do not write graphics files** checked) to get the src attributes right.

3.7.5 Choosing postprocessing options

Mif2Go can do the following after converting your document:

[Update WinHelp TOC and IX](#)

[Compile WinHelp or HTML Help](#)

[Assemble and archive deliverables](#)

*Update WinHelp
TOC and IX*

To update an existing WinHelp contents file when you are converting a single file for a multi-file project, on the **Mif2Go Export** dialog check:

Make combined TOC/IX

The book containing the file you are converting must be open in FrameMaker. If you are converting a book rather than a single file, this option is disabled, and the contents file is

always updated. See §8.2.8 [Setting basic WinHelp options in the configuration file](#) on page 248.

*Compile WinHelp
or HTML Help*

If you are generating WinHelp or MS HTML Help and you specified a Help project file for your conversion project, on the **Mif2Go Export** dialog you can check:

Compile Help

to make **Mif2Go** run the appropriate Help compiler after conversion. Usually it is safe to use this option for MS HTML Help. However, if you are converting to WinHelp, ***use this option only for small projects***. Windows might run out of memory during compilation; or, your graphics might not get included in the Help file; or, the Help compiler might run out of memory. For most WinHelp projects you should leave this option unchecked, and compile from Help Workshop instead. See §7.2.4 [Compiling and distributing Help systems](#) on page 204.

*Assemble and
archive
deliverables*

If your configuration file includes options for assembling and archiving conversion results, when you check:

Wrap and Ship

on the **Mif2Go Export** dialog, **Mif2Go** executes the specified commands and puts the results in the specified directory(ies). See §35 [Producing deliverable results](#) on page 955.

4 Editing configuration files

This section explains how **Mif2Go** configuration files are created, and presents the rules for adding and modifying configuration settings. Topics include:

- §4.1 [Working with Mif2Go configuration files](#) on page 91
- §4.2 [Editing files with the Configuration Manager](#) on page 91
- §4.3 [Understanding where project settings come from](#) on page 102
- §4.4 [Understanding the rules for configuration settings](#) on page 102
- §4.5 [Specifying file paths in configuration settings](#) on page 105
- §4.6 [Using wildcards in configuration settings](#) on page 106
- §4.7 [Commenting out configuration sections](#) on page 107
- §4.8 [Ending a configuration file](#) on page 107

See also:

- §30 [Working with templates](#) on page 849

4.1 Working with Mif2Go configuration files

To add or change conversion settings after you set up a **Mif2Go** project, you must edit the contents of one or more *configuration files*: text files with file extension `.ini`.

You will need two tools to work effectively with **Mif2Go** configuration files:

- The **Mif2Go Configuration Manager**, to see and manipulate individual sections and settings selected from all relevant configuration files and system templates that apply to your project; see §4.2 [Editing files with the Configuration Manager](#) on page 91
- A **text editor** (even Notepad) to inspect and optionally edit all sections and settings in individual configuration files:
 - Make sure you use only ANSI, or UTF-8, encoding; do not use UTF-16 for configuration files.
 - Do not use Word, or any other application that gets an exclusive-write lock on files.

You might find it useful to have both tools open at once, so you can readily see in context, in the text editor, any changes you make with the Configuration Manager.

4.2 Editing files with the Configuration Manager

The **Mif2Go** Configuration Manager gives you access to all the configuration settings and macros that affect a conversion project, regardless of which configuration or template files hold those values.

In this section:

- §4.2.1 [Understanding how to use the Configuration Manager](#) on page 92
- §4.2.2 [Starting the Configuration Manager](#) on page 93
- §4.2.3 [Setting Configuration Manager preferences](#) on page 93
- §4.2.4 [Establishing a starting point](#) on page 95
- §4.2.5 [Choosing a configuration category or file type](#) on page 95
- §4.2.6 [Understanding variable vs. fixed names and keys](#) on page 95
- §4.2.7 [Choosing the kind of change to make](#) on page 96

- §4.2.8 [Selecting a configuration section](#) on page 100
- §4.2.9 [Selecting a configuration setting](#) on page 100
- §4.2.10 [Selecting a configuration file](#) on page 101
- §4.2.11 [Specifying a final value](#) on page 101

4.2.1 Understanding how to use the Configuration Manager

With the Configuration Manager you can “drill down” to see where a setting is located, and also see the scope of each setting: does it affect only the current project, all your projects, all outputs of one type, or perhaps only one document?

In this section:

- §4.2.1.1 [Drilling down to find a section or setting](#) on page 92
- §4.2.1.2 [Heeding status and result messages](#) on page 92
- §4.2.1.3 [Getting help with controls and configuration data](#) on page 93
- §4.2.1.4 [Providing help for your own macro definitions](#) on page 93
- §4.2.1.5 [Correcting configuration errors](#) on page 93

4.2.1.1 Drilling down to find a section or setting

The process goes like this:

1. Select a project (thus identifying the project configuration file), or designate another configuration file; all other files or templates referenced from that initial file via [Templates] settings can be included in the current Configuration Manager session.
2. Decide what kinds of settings you want to work with: general configuration settings macros, or content models.
3. Pick what you want to do: add, change, delete, restore, copy, move, or merge a setting or a whole section.
4. Pick the section (and possibly setting) to work on. At this point you can see where the item occurs in all the configuration files included in the current session; and you see the scope of effect of each value.
5. Pick the file where you want the change to take place.
6. Apply your selections to make the change.

4.2.1.2 Heeding status and result messages

A message box at the bottom of each Configuration Manager page displays status messages and reports the result of each action. If the Configuration Manager is unable to proceed on a retry of the same action, you will need to exit and restart the program.

For example, if you try to use the Configuration Manager to make changes in a file that is open in another application that locked the file, you will see this message:

```
File not updated, changes in .new
```

This means the Configuration Manager was unable to write your changes to that file, and instead made a copy of the file but with extension `.new`, made the changes there, and saved the `.new` file in the same directory as the original. If this happens, you must exit the Configuration Manager and release the file from the application that locked it. Then restart the Configuration Manager and retry the action.

4.2.1.3 Getting help with controls and configuration data

On any Configuration Manager page, after making a selection or moving focus to a control, you can click [?] in the lower right corner of the page to open a section of the **Mif2Go User's Guide** that explains that selection or control.

Note: Make sure that `ugmif2go.chm`, the HTML Help version of the **Mif2Go User's Guide**, is available in `%OMSYSHOME%\m2g\usersguide`; see §1.3.3.2 [Finish installing Mif2Go](#) on page 57.

4.2.1.4 Providing help for your own macro definitions

The Configuration Manager displays one-line descriptions of all fixed-name configuration sections, including all macro definitions supplied in the **Mif2Go** distribution.

For definitions of macros that you create and name, by default the Configuration Manager displays:

No Help for this section

To provide descriptive help for a macro, prefix keyword `help` with special comment delimiters `;` = to keep it from being parsed as part of the macro. For example:

```
[AnotherURL]
;=help = The URL for the DTS project at SourceForge
http://sourceforge.net/projects/ditatestsuite
```

4.2.1.5 Correcting configuration errors

If any of the configuration files open for a given session contain settings that appear not to be valid in their section, or sections that appear not to be valid in their file, the Configuration Manager displays the names of those settings or sections in red. You will need to delete the items, rename them, or move them to a valid location.

4.2.2 Starting the Configuration Manager

To access configuration values for your project, start the **Mif2Go** Configuration Manager, either of the following ways:

- **On the Windows desktop:** double-click the shortcut to `%OMSYSHOME%\common\bin\m2gcm.exe`. (If you do not already have a desktop shortcut to this program, see §1.3.3.2 [Finish installing Mif2Go](#) on page 57.) The Configuration Manager opens to the **Start** page; see §4.2.4 [Establishing a starting point](#) on page 95.
- **From the Choose Project dialog in FrameMaker;** see §3.3 [Creating a Mif2Go conversion project](#) on page 78.

Before you use the Configuration Manager to change the way your projects work, consider visiting the **Preferences** page; see §4.2.3 [Setting Configuration Manager preferences](#) on page 93.

4.2.3 Setting Configuration Manager preferences

You can change text colors in Configuration Manager displays, and choose to have all your configuration edits annotated and timestamped in configuration files.

In this section:

- §4.2.3.1 [Specifying colors for different types of settings](#) on page 94
- §4.2.3.2 [Annotating changes made to configuration files](#) on page 94

4.2.3.1 Specifying colors for different types of settings

These are the colors the Configuration Manager uses to display different kinds of items:

Available fixed-name sections	Configuration sections that are valid in the category of settings you are working with, but that do not appear in any files for the current session.
Available fixed-key settings	Configuration settings that are valid in the section you are working with, but that do not appear in any instances of that section in the files for the current session.
Variable names and keys	Configuration sections such as format and macro definitions, and settings with keys such as format names or object identifiers, whether or not they appear in the files for the current session.
Internal defaults	Values that Mif2Go uses for settings that do not appear in any of the files for the current session.
System configuration files	Values specified for settings in the system configuration templates. You cannot change these values; instead, you override them in the corresponding local configuration templates. See §30.5.1 Understanding what configuration files are available on page 857.

To change the color of an item, click the colored box next to the description. The Windows color picker opens. Here you can go wild with any colors that float your boat; however, avoid red. The Configuration Manager uses red to flag invalid section and setting names; see §4.2.1.5 [Correcting configuration errors](#) on page 93.

Click **OK** to dismiss the color picker and establish the new color. The color change takes effect immediately.

4.2.3.2 Annotating changes made to configuration files

To keep an annotated record of all the changes made to your files with the Configuration Manager, check **Include history comments**.

To own up to these changes, under **User name for history comments** provide an identifier such as your name or initials.

The Configuration Manager inserts a comment above each change, showing what was changed and when. Deleted items are “commented out”, rather than removed. For example:

```
[HTMLOptions]
Title=Mif2Go User's Guide
;2012-11-30 15:16:08: CS deleted duplicate Title
;=Title=Mif2Go User's Guide
```

Annotation takes effect immediately.

4.2.4 Establishing a starting point

On the Configuration Manager **Start** page, select one of your conversion projects, or choose a configuration template or file to start with. If you select a project, the project configuration file for that project becomes the starting point for the current session.

Note: To get your project into the list of available projects on the Configuration Manager **Start** page, choose **Configure** on the *Choose Project* dialog in FrameMaker; see §3.3 [Creating a Mif2Go conversion project](#) on page 78.

The file designated as a “starting” configuration file determines which other configuration files the Configuration Manager can include in your current session. Candidates include all configuration files referenced through the chain of [Templates] settings (see §30.2 [Referencing configuration files and templates](#) on page 851) in the starting configuration file, and in the files referenced by that file. Which subset of these files will be accessed depends on which category of settings you intend to work with.

Once you have selected a starting point, click **Apply file**; the Configuration Manager switches to the **Category** page. See §4.2.5 [Choosing a configuration category or file type](#) on page 95.

4.2.5 Choosing a configuration category or file type

On the Configuration Manager **Category** page, choose a category of settings to work with, or a type of configuration file. You are not locked into your choice; you can always return to this page to switch to a different category or configuration type.

At the top of the Configuration Manager **Category** page you see the full path to the starting configuration file for the current session; this is either the project configuration file for a project you selected, or another file you specified on the **Start** page; see §4.2.4 [Establishing a starting point](#) on page 95.

*Select type of file
or category of
settings*

Under **Select type of configuration file**, you can highlight the kind of configuration file or template you want to focus on. As an alternative, under **Select category**, click one of the following categories:

<u>Category</u>	<u>Description</u>	<u>Ref.</u>
General Configuration	Project configuration options and settings	5
Macros	Definitions of Mif2Go macros	28
Content Models	Configuration-style representation of a DTD	32

If the category has a + in front of it, click the + to see a list of subcategories.

*Section matches
and Setting
matches*

To narrow down your selection, you can specify the name of a configuration section, the name of a setting in that section, or both; and you can use wildcards in either name. If you specify both a section name and a setting name, make sure that setting actually is valid in the named section.

Apply selections

Once you have selected a category or file type (and optionally specified section and/or setting names), click **Apply selections**; the Configuration Manager switches to the **Action** page. See §4.2.7 [Choosing the kind of change to make](#) on page 96.

4.2.6 Understanding variable vs. fixed names and keys

To specify certain actions for the Configuration Manager to perform, you must distinguish between variable-name and fixed-name configuration sections, and between variable-key and fixed-key settings.

*Variable-name vs.
fixed-name
sections*

- A **variable-name configuration section** can have any name at all; examples are macro definitions, where the section name is the name of the macro.
- A **fixed-name configuration section** has a name defined by Mif2Go; examples include Setup and HTMLOptions.

*Variable-key vs.
fixed-key settings*

- **Variable-key settings** are characterized by keys that usually consist of an object identifier (such as the settings in section GraphGroup).
- **Fixed-key settings** must use a key from a set of Mif2Go-specified names for keys that are valid in their section (such as the settings in section Setup). See §33.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 923.

4.2.7 Choosing the kind of change to make

On the Configuration Manager **Action** page, select the kind of change you want to make to a configuration. Some actions distinguish between variable-name and fixed-name sections, or between variable-key and fixed-key settings; to determine which to select for the section or setting you want to change, see §33.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 923.

Under **Select action to be performed**, click a button to act on a section or a setting. Once you have selected an action, click **Apply action**. Provided there are no duplicate settings or sections in any of the files for the current session, the Configuration Manager switches to one of the following:

- the **Section** page, to select a configuration section
- the **Setting** page, if only one section applies
- the **.ini file** page, if both section and setting are already determined.

*Duplicate
sections or
settings*

However, if the Configuration Manager finds duplicate settings in a section or duplicate sections in a file, instead of proceeding with the action you specified, the Configuration Manager changes your selection to one of the merge options; see §4.2.7.8 [Merging duplicate sections or settings](#) on page 100. You can change it back again, but the Configuration Manager will continue to nag you about duplicates until you resolve them.

In this section:

- §4.2.7.1 [Adding a new section or setting](#) on page 96
- §4.2.7.2 [Editing a section or setting](#) on page 97
- §4.2.7.3 [Deleting a section or setting](#) on page 97
- §4.2.7.4 [Restoring a deleted section or setting](#) on page 98
- §4.2.7.5 [Renaming a section or setting](#) on page 98
- §4.2.7.6 [Moving a section or setting](#) on page 99
- §4.2.7.7 [Copying a section or setting to another configuration file](#) on page 99
- §4.2.7.8 [Merging duplicate sections or settings](#) on page 100

4.2.7.1 Adding a new section or setting

You can add a new section to a selected configuration file, or a new setting to a section. Under **ADD new item** on the Configuration Manager **Action** page, choose one of the following:

Add new fixed-name section to file

A fixed-name section is pretty much any configuration section that is neither a format definition nor a macro definition. You select from applicable section names on the **Section** page.

Add new variable-name section to file	To add a macro definition, select this option. You get to specify the name on the Section page.
Add new setting to section	If you try to add a setting to a file that does not contain the section that the setting belongs in, the Configuration Manager creates that section for you. <i>Do not use for macro definitions.</i>
Add new variable-key setting to one section	You get to choose the configuration section where you want the new setting. <i>Do not use for macro definitions.</i>

If you try to add a section to a file that already has a section by the same name, the Configuration Manager presents the existing section for you to edit instead; see §4.2.7.2 [Editing a section or setting](#) on page 97.

Note: You cannot add a setting to a named macro. Macro sections do not contain settings. Instead, choose **Edit full section content**; see §4.2.7.2 [Editing a section or setting](#) on page 97.

Click **Apply action**; the Configuration Manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

4.2.7.2 Editing a section or setting

You can edit an entire section, or a single setting, in a selected configuration file. Under **EDIT existing item** on the Configuration Manager **Action** page, choose one of the following:

Edit full section content	Edit the content of a section, and also the section heading and any comments; you can even change the name of the section.
Edit one setting in section	First you choose a section, then you pick the setting, then you choose the file where you want to make changes. <i>Do not use for named macros</i> ; instead, edit the full section.

If you edit the value for a setting, and you want a leading space before the value, you have to add that space explicitly. A single space is stripped; multiple spaces are preserved. If you change something else, spaces in the value are not altered, unlike spaces before the equals sign, which are always removed.

Click **Apply action**; the Configuration manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

4.2.7.3 Deleting a section or setting

You can delete an entire section, or a single setting, from a selected configuration file. Under **DELETE existing item** on the Configuration Manager **Action** page, choose one of the following:

Delete section from one file	When you choose to delete a section, the Configuration Manager allows you to inspect and optionally edit the content of the section before committing to its deletion.
-------------------------------------	--

Delete setting from one section

First you choose a section, then you pick the setting, then you choose the file from which you want to delete the setting. *Do not use for named macros*; macro sections do not contain settings.

When you delete a section or a setting, the Configuration Manager does not remove the item from the file, but instead deactivates it by commenting it out. This allows you to restore the item later. If you really want the item expunged leaving no trace, the Configuration Manager allows you to edit it; at that point you can simply erase the item.

Click **Apply action**; the Configuration manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

Note: The Configuration Manager will continue to show the deleted item until you return to the **Action** or **Category** page and make another selection.

4.2.7.4 Restoring a deleted section or setting

You can restore an entire deleted section, or a single setting, in a selected configuration file. Under **RESTORE deleted item** on the Configuration Manager **Action** page, choose one of the following:

Restore deleted section in one file

You can restore a section that has been marked for deletion. The Configuration Manager will remove the commenting that deactivated the section. To restore a section that was physically removed from a file, instead add it (§4.2.7.1 [Adding a new section or setting](#) on page 96) or copy it from another file (§4.2.7.7 [Copying a section or setting to another configuration file](#) on page 99).

Restore deleted setting in one section

First you choose a section, then you pick the setting, then you choose the file where you want to restore the setting.

When you delete a section or a setting, the Configuration Manager does not remove the item from the file, but instead deactivates it by commenting it out. When you choose to restore an item, the Configuration Manager removes the commenting that deactivated the item.

Click **Apply action**; the Configuration manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

4.2.7.5 Renaming a section or setting

You can rename a variable-name section, or rename a single variable-key setting, in a selected configuration file. Under **RENAME existing item** on the Configuration Manager **Action** page, choose one of the following:

Rename variable-name section in one file

Only variable-name sections, such as macro definitions, can be renamed.

Rename variable-key setting in one section

Only variable-key settings can be renamed, with keys that represent format names or object group names. First you choose a section, then you pick the setting, then you choose the file where you want to change the name of the setting.

If a file contains the wrong fixed-name section, you have to delete that section and add the correct section. On deletion you might get away with simply changing the section name; this could work only if the settings already present are valid under the new name.

Click **Apply action**; the Configuration manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

4.2.7.6 Moving a section or setting

You can move sections within or between files, and you can move settings either within a section or between instances of that section in different files. Under **MOVE existing item** on the Configuration Manager **Action** page, choose one of the following:

Move section within one file	For the most part, the order of sections within a configuration file does not affect functionality. However, you might want to change the order of sections in a file for readability.
Move section between files	You might want to move a section “upstream” or “downstream”; that is, make it apply more widely or less widely.
Move setting within one section	The only time the order of settings within a section affects functionality is when you use wildcards in the names of variable-key settings, such as in group names for graphics or tables. This option lets you move a selected setting up or down within the same section.
Move setting between files	To change the scope of a setting, you can move it “upstream” (to a configuration file with a wider scope) or “downstream” (to a configuration file with a narrower scope).

Click **Apply action**; the Configuration manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

4.2.7.7 Copying a section or setting to another configuration file

You can copy sections between files, and you can copy settings between instances of the same section in different files. Under **COPY existing item** on the Configuration Manager **Action** page, choose one of the following:

Copy section between files	Insert a full copy of the designated section in another file, even if that section does not exist in the “from” file.
Copy setting between files	Copy one setting from a section you select to an instance of the same section in another file. If the section is not already present in the destination file, the Configuration Manager creates it there before copying the setting.

Click **Apply action**; the Configuration manager switches to the **Section** page, the **Setting** page, or the **.ini file** page, depending on the action and the available sections or settings.

4.2.7.8 Merging duplicate sections or settings

The Configuration Manager detects duplicate sections in a file, and duplicate settings in a section, and warns you about them on the **Action** page; inviting you to fix them by selecting for you, under **MERGE duplicate items**, one of the following:

Merge duplicate sections in one file

The Configuration Manager moves the second instance of the duplicate section to a position immediately after the first, and comments out the second section heading. The effect is to include all settings from both sections in the first section, possibly resulting in duplicate settings. You get a chance to edit the merged section.

Merge duplicate settings in one section

The Configuration manager edits the first of the duplicated settings to show the value you select on the **Finish** page, then comments out the second setting, even if the second was the one that had the correct value. This is because only the first of duplicate settings in a section has any effect.

Click **Apply action**; the Configuration manager switches to the **.ini file** page, where the offending section name or setting value shows in the **Value** column for the highlighted file that contains the duplicates.

4.2.8 Selecting a configuration section

At the top of the Configuration Manager **Section** page you see the full path to the starting configuration file for the current session, and also the edit action you selected on the **Action** page; see §4.2.7 [Choosing the kind of change to make](#) on page 96.

Under **Select section for action** you see listed all the sections (if any) that meet the criteria you established on the **Category** page.

Click a section name to select it. Immediately below the selection box you see a statement that characterizes the section.

If you had chosen to add a new variable-name section on the **Action** page, you could specify its name after **Name for new section**. Or, if you had chosen to rename a variable-name section on the **Action** page, you could specify its new name after **Rename section**.

Once you have selected a section, or possibly named a new section or renamed an existing section, click **Apply section**; what happens next depends on whether the action you chose pertains to a section or to a setting:

Section The Configuration Manager switches to the **.ini File** page; see §4.2.10 [Selecting a configuration file](#) on page 101

Setting The Configuration Manager switches to the **Setting** page; see §4.2.9 [Selecting a configuration setting](#) on page 100.

4.2.9 Selecting a configuration setting

At the top of the Configuration Manager **Setting** page you see the full path to the starting configuration file for the current session, and also the edit action you selected on the **Action** page; see §4.2.7 [Choosing the kind of change to make](#) on page 96. Next you see the name of the section you selected, and a note of its purpose.

Under **Select setting for action** you see listed all the settings (if any) that meet the criteria you established on the **Category** page and that are valid (or already exist) in the section you selected.

If you chose to add a fixed-key setting, those listed in black are already in use in at least one configuration file for the current session, while those listed in an alternate color (see §4.2.3 [Setting Configuration Manager preferences](#) on page 93) are available for use.

If you chose to add a variable-key setting, you can specify **Name for new setting**.

If you chose to edit a setting, only those currently in use are listed.

If you chose to rename a variable-key setting, you can **Rename setting**.

Once you have selected a setting or entered a new setting name, click **Apply setting**; the Configuration manager switches to the **.ini File** page, where you choose the file in which you want to make the change; see §4.2.10 [Selecting a configuration file](#) on page 101.

4.2.10 Selecting a configuration file

At the top of the Configuration Manager **.ini File** page you see the full path to the starting configuration file for the current session, and also the action you selected on the **Action** page; see §4.2.7 [Choosing the kind of change to make](#) on page 96. Next you see the name of the section you selected, and a note of its purpose; then, if you are changing a setting, the name and value of the setting you selected, with a note of its purpose.

Under **Select .ini to change for this action** you see listed all the configuration files and templates that apply to the current session, with the value of the setting in each file (or the name of the section, if present in the file), along with a statement of the scope of effect of the value. If the setting in question has an internal default value (see §4.2.3 [Setting Configuration Manager preferences](#) on page 93), that value is listed at the top.

The configuration files for which the category and section you selected are valid, are listed from greatest scope of effect (at top) to least scope (at bottom). A setting in a particular configuration file overrides the value of the same setting in all configuration files listed above, and is overridden by the value in any configuration files listed below.

Once you have selected the file where you want to make the change to the setting or section, click **Apply ini file**; the Configuration manager switches to the **Finish** page, where you finally get to execute the action you chose; see §4.2.11 [Specifying a final value](#) on page 101.

4.2.11 Specifying a final value

At the top of the Configuration Manager **Finish** page you see the full path to the starting configuration file for the current session, and also the action you selected on the **Action** page; see §4.2.7 [Choosing the kind of change to make](#) on page 96. Next you see the name of the section you selected, and a note of its purpose; then, if you are changing a setting, the name and value of the setting you selected, with a note of its purpose. Next you see the name of the configuration file you selected, then the “current value” of the setting in that file, and the purpose of the value.

The rest of the **Finish** page is devoted to giving you values to select from or settings or sections to edit, depending on the action.

If you edit the value for a setting, and you want a leading space before the value, you have to add that space explicitly. A single space is stripped; multiple spaces are preserved. If

you change something else, spaces in the value are not altered, unlike spaces before the equals sign, which are always removed.

Once you have completed your edits and/or selections, click **Finish action**; the Configuration Manager executes your changes and then switches back to the **Action** page; see §4.2.7 [Choosing the kind of change to make](#) on page 96. There you can choose another action, or you can go back to the **Start** or **Category** page to establish a different universe of discourse.

To see what actually happened, either inspect the file you selected in a text editor, or choose an **Edit** action on the **Action** page, and select the relevant section and file.

4.3 Understanding where project settings come from

When you set up a new conversion project, the **Mif2Go** plug-in copies a new output-type-specific starting configuration file into your project directory. This file is populated with the settings you specify in the *Set Up* dialog (see §3.4 [Choosing project set-up options](#) on page 79). **Mif2Go** gets this file from a repository of configuration templates located in your **Mif2Go** distribution; see §30.1.1 [Understanding how templates are organized](#) on page 849. Each configuration template already contains values for basic settings specific to the output type for your project.

*Default
configuration
values*

Configuration values presented in the *Set Up* dialog (see §3.4 [Choosing project set-up options](#) on page 79) are not always the same as the internal default values for configuration settings:

- Set-up value:* The value people usually want (or expect) for a new project for a given output type.
- Internal default:* The value **Mif2Go** applies when the setting is missing entirely from the configuration files for your project.

Often, the internal default value produces the effect you would have experienced before a feature was added to **Mif2Go**; this is to maintain backward compatibility with existing configuration files. The effect is almost always equivalent to turning the feature “Off”. However, if the feature corrects a defect, the corresponding configuration value might default to “On”, with the setting provided to support users who had already put a workaround in place and wanted its functionality left alone.

*Referenced
configuration
values*

Your project configuration file includes references to:

- an optional document-specific configuration file created when you set up this or a previous project (see §3.5 [Understanding how Mif2Go sets up a project](#) on page 82)
- a required chain of configuration templates located in your **Mif2Go** distribution.

Your project incorporates by reference any settings those files contain, unless the settings are overridden in your project configuration file. If you intend to work with many conversion projects, you might want to inspect and possibly modify the local editions of some of these templates. See §30.1 [Working with configuration templates](#) on page 849.

4.4 Understanding the rules for configuration settings

Every **Mif2Go** configuration file must begin with at least one line of header text, even if it is an empty line. The content of the line does not matter to **Mif2Go**, as long as it does not duplicate the name of a configuration section.

After the header text, each configuration file contains a series of *sections*. Each section consists of a section name in square brackets, followed by a list of *settings* of the form *Key=Value* or *Key=Command*, each on a separate line; and possibly by one or more *comments*:

```
[Section]
Key = Value
Key = Value1 Value2 Value3 ...
Key = Command
; Comment
```

Section names may not contain spaces or punctuation. The opening bracket for each section name must be in column 1.

Keep in mind these Microsoft rules for configuration files:

- **Section names must be unique;** if there are duplicate section names in a configuration file, only the first instance is processed.
- **Key names must be unique in a section;** if you repeat a key name in the same section, only the first instance is processed.

And these **Mif2Go** rules:

- **The first line in the file must be a comment;** **Mif2Go** requires a header line.
- **No more than one space after the equals sign;** otherwise, *bad things can happen*:
 - If the value is Boolean (Yes/No), **Mif2Go** treats it as No, even if you typed Yes.
 - If the value is a string, all spaces after the first are included in the string.

Consider all of the following:

Section names must be unique
 Key names must be unique in a section
 Key names must be valid ASCII
 Key names are not case sensitive, by default
 Fixed-key sections differ from variable-key sections
 Order of settings can be important for variable keys
 Formats must be in catalogs
 Multiple values are separated by spaces
 Spaces and tabs: some retained, some removed
 Comments start with a semicolon
 Boolean values can be expressed various ways

Section names must be unique

Section names must be unique. If you use the same section name twice in your configuration file, *only the first section is processed*. Otherwise, order of sections does not matter, except for macro sections (see §28.1.1.2 [Understanding where you can define named macros](#) on page 788).

Key names must be unique in a section

Each *key=* setting in a given section must be the only setting for that key in that section. A common error is to add a setting to a section that already has a setting for that key. For example, any repeated lines assigning additional values to the same format name are ignored; only the first line is processed. Instead, place any additional values on the same line as the first, separated by spaces.

Key names must be valid ASCII

All ASCII characters are valid in key names, with the following exceptions:

- “?” and “*” are treated as wildcards, unless you turn off wildcard usage; see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113. (However, when you override a configuration setting with a configuration variable, **Mif2Go** does not recognize wildcards in the key name; see §33.2.4 [Assigning values to configuration variables](#) on page 922.)

- “;” or “[” must be prefixed with escape character “\” if you want to *start* a key name with either of these characters.

Spaces are nominally allowed in key names, but the spaces are ignored unless you turn off [Options]SpacelessMatch; see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113. Do not use spaces if you can possibly avoid them.

Key names are
not case
sensitive, by
default

Comparisons of key names are caseless, unless you turn on case sensitivity; see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113. (However, when you override a configuration setting with a configuration variable, the key name *is* case sensitive; see §33.2.4 [Assigning values to configuration variables](#) on page 922.)

Fixed-key
sections differ
from variable-key
sections

Configuration files contain two kinds of sections: those with *fixed keys* (key names predefined by **Mif2Go**) and those with *variable keys*. For example, sections such as [HTMLOptions] and [WordOptions] are for settings with fixed key names; sections such as [HTMLParaStyles] and [HelpStyles] are for settings with key names you specify, typically names of FrameMaker formats.

Order of settings
can be important
for variable keys

In a fixed-key section, the order of settings does not matter. Order is important only in sections where you can use variable keys, and usually only if you use wildcards in key names (see §4.6 [Using wildcards in configuration settings](#) on page 106). However, there are exceptions; for example, see §29.4.2 [Observing restrictions on redefining marker behavior](#) on page 840.

Formats must be
in catalogs

Often the variable-key names you specify are names of formats in your FrameMaker document, such as paragraph, character, or table formats. Make sure each format you use for a key name actually appears in the appropriate catalog in FrameMaker; **Mif2Go** cannot process formats that are not in a FrameMaker catalog.

Multiple values
are separated by
spaces

Some variable-key sections allow multiple values for each key: sections such as [HTMLParaStyles], [WordStyles], and [HelpStyles], where you can assign multiple properties to each FrameMaker format. Use spaces between values.

Spaces and tabs:
some retained,
some removed

Mif2Go treats spaces and tabs in configuration settings as follows:

- Spaces and tabs before the *Key* and before the equals sign are ignored, unless [Options]SpacelessMatch=No, in which case those before the *Key* are *not* ignored (see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113).
- If the equals sign is followed by one or more spaces or tabs, the first such space or tab is removed, and *the rest are treated as part of the value*. Put no more than one space after the equals sign. If you want to align settings vertically for readability, put extra spaces *before* the equals sign, not after.
- All spaces and tabs that follow a value are retained in the output.
- Do not try to indent settings in your project configuration file. When **Mif2Go** updates this file, Windows rewrites the file, and deletes all leading spaces in the settings. You can use indentation in macro definitions in other configuration files and macro libraries.

Comments start
with a semicolon

Lines that start with a semicolon “;” are comments. For a line to be treated as a comment, the semicolon must be the *first* character on the line (no leading blanks or tabs). **Mif2Go** pays no attention to comment lines; you can use them to add your own notes. However, *do not try to “comment out” a section* by inserting a “;” in front of the section name; all settings that follow such a line, up to the next line that starts with a “[”, would be added to the settings for the preceding section. To comment out a section, see §4.7 [Commenting out configuration sections](#) on page 107.

Boolean values
can be expressed
various ways

For an On/Off value, **Mif2Go** recognizes “1” (numeral one), “Yes”, and “True” as On, and “0” (zero), “No”, and “False” as Off.

4.5 Specifying file paths in configuration settings

Path and file names should conform to the requirements listed in §1.1.2 [File, directory, and path names](#) on page 51, otherwise you risk run-time errors. Additional considerations:

Path separator can be “\” or “/” (except sometimes!)

Paths that contain spaces must be quoted

Most relative paths relate to the project directory

Paths to configuration templates should be absolute

Some relative paths relate to the configuration file location

Paths to other applications must be absolute.

*Path separator
can be “\” or “/”
(except
sometimes!)*

When you specify a file path in a configuration setting, you can use either a backslash “\” or a forward slash “/” as a separator character. A forward slash is preferred, except in the following cases, where you *must* use a backslash:

- Windows system commands; see §34.4 [Executing operating-system commands](#) on page 937
- Windows command parameters; for example, see §35.11 [Archiving deliverables](#) on page 973.

In FrameMaker dialogs, the backslash has a special meaning that trashes paths beyond recovery. In Windows API calls, forward slashes work fine, because the original Windows programmers compiled Windows on VAX/VMS machines.

*Paths that contain
spaces must be
quoted*

If a path contains any spaces, enclose the entire path in quotes. See §1.1.2 [File, directory, and path names](#) on page 51 and §3.1 [Checking set-up and conversion requirements](#) on page 77.

*Most relative
paths relate to the
project directory*

Most path settings (other than those listed in [Table 4-1](#) on page 106) can be either relative or absolute; however, file paths you specify via the FrameMaker plug-in; that is, in a FrameMaker dialog box (as opposed to a setting in a configuration file); must be absolute. Relative file paths can make your conversion project portable. Many support issues arise when a project is moved, after which some buried links stop working.

When you specify a relative file path in a configuration setting, the path is relative to the project directory, with the following exceptions:

- settings listed in [Table 4-1](#) on page 106
- [Automation]ShipPath, which is relative to the wrap directory; see §35.3 [Understanding path values for deliverables](#) on page 957.

*Paths to
configuration
templates should
be absolute*

Settings that reference configuration templates, or other files in the **Mif2Go** distribution directory structure, should use absolute paths that begin with environment variable %OMSYSHOME%; for example:

```
Configs = %omsysHOME%\m2g\local\config\local_m2htm_config.ini
```

See §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54.

*Some relative
paths relate to the
configuration file
location*

If you specify a relative path in any of the settings listed in [Table 4-1](#) on page 106, the path is considered to be relative to the location of the configuration file in which the setting occurs. This means that if you move such a setting from one configuration file to another at a different level in your project directory structure, *the path will no longer be correct*.

If you want project portability, the price is using a fixed directory structure, where both the _config directory (see §30.3.3 [Deciding where to keep document-specific configuration files](#) on page 854) and the project directory are immediately below the source directory. If you have files all over the place, portability becomes impossible.

Table 4-1: Absolute vs. relative file-path settings

Section	Setting	Absolute or relative file path?	Ref.
[Templates]	<i>All settings</i>	Paths to templates in the Mif2Go distribution should be absolute and start with %OMSYSHOME%	30.1
		Paths to your own template files should be relative to the location of the configuration file in which the setting occurs	30.6
[Setup]	IDFileName	Absolute path recommended	5.3.4
	PrjFileName	Absolute path recommended	C.3
	CheckLinkLog	Absolute path recommended	5.1.5
	ConfigTemplate	<i>deprecated</i>	30.2
	TemplateFileName	Absolute path recommended	
[Logging]	LogFileName	Keep setting in project configuration file	5.2
	HistoryFileName	Keep setting in project configuration file	5.2
[OmniHelpOptions]	ProjectTemplate	Keep setting in project configuration file	10.5.7

Paths to other applications must be absolute

File paths to non-**Mif2Go** executables must be absolute. However, a better way would be to make sure those executables are on your system PATH, so your conversion project is portable.

4.6 Using wildcards in configuration settings

In a variable-key setting, you can apply the same value to multiple keys by substituting a wildcard “*” or “?” for all or part of the key name, as follows:

- A question mark can appear anywhere in a key name, substituting for any one character; multiple question marks substitute for the same number of characters.
- An asterisk can appear only at the end of a key name, substituting for one or more characters.

You can use wildcards whenever the key is a format name or an identifier, provided you have not turned off wildcard usage (see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113). For example, to make all FrameMaker paragraphs whose format names start with *Heading* appear bold and centered in HTML output:

```
[HTMLParaStyles]
Heading*=Bold Center
```

You can exclude one or more key names from a group by listing the exceptions first:

```
[HTMLParaStyles]
Heading4=
Heading3=Bold Left
Heading*=Bold Center
z????title=Bold Right
*=Left
```

In this example:

- No HTML style properties would apply to *Heading4* paragraphs.
- *Heading3* paragraphs would appear left-aligned and bold in HTML.
- All remaining *Heading** paragraphs would be centered and bold.
- All paragraphs whose format names start with z, followed by any four characters, and end with *title*, would be right-aligned and bold.
- Paragraphs in all other formats would be left-aligned.

Mif2Go applies the first entry in a section that matches for each key name, so always put the exceptions before the general case.

4.7 Commenting out configuration sections

If you need to comment out an entire section in your project configuration file, perhaps to test an alternative approach, you can place a semicolon at the beginning of each setting in that section. An easier way is to place a semicolon after the opening bracket of the section head; for example:

```
[ ;Templates]
```

This has the effect of giving that section a name that has no meaning to **Mif2Go**, so the settings for that section will be ignored.

Note: If you place the semicolon *before* the opening bracket, the settings will become part of the previous configuration section, rarely what you want.

4.8 Ending a configuration file

All configuration files and templates in your **Mif2Go** distribution end with a dummy section that signifies no more settings:

```
[ End]
```

If you create additional configuration files or templates, end them with this section.

5 Setting basic conversion options

This section explains how to use basic **Mif2Go** configuration settings to convert documents from FrameMaker to other representations. Topics include:

- §5.1 [Specifying operating settings](#) on page 109
- §5.2 [Logging conversion events](#) on page 115
- §5.3 [Identifying files and objects](#) on page 117
- §5.4 [Applying FrameMaker conditions and variables](#) on page 122
- §5.5 [Converting FrameMaker-generated files](#) on page 124
- §5.6 [Generating/updating before converting](#) on page 126
- §5.7 [Processing graphics](#) on page 126
- §5.8 [Converting structured documents](#) on page 135
- §5.9 [Converting equations](#) on page 136
- §5.10 [Creating hotspots for hypertext links](#) on page 138
- §5.11 [Repurposing FrameMaker markers](#) on page 139

Print RTF For additional settings specific to print RTF, see:

- §6 [Converting to print RTF](#) on page 141

Help systems For additional settings specific to on-line Help, see:

- §7 [Producing on-line Help](#) on page 199 *through*
- §12 [Generating Eclipse Help](#) on page 403

HTML, XML For additional settings specific to HTML and XML, see:

- §13 [Converting to HTML/XHTML](#) on page 423 *through*
- §27 [Marking HTML table cells for WAI](#) on page 775

Graphics For additional settings specific to graphics conversion, see:

- §31 [Working with graphics](#) on page 869

5.1 Specifying operating settings

In this section:

- §5.1.1 [Checking output type and file extension](#) on page 110
- §5.1.2 [Excluding files from book conversions](#) on page 110
- §5.1.3 [Reusing or discarding MIF files](#) on page 111
- §5.1.4 [Reusing or discarding ASCII DCL files](#) on page 111
- §5.1.5 [Checking for broken links in HTML or XML output](#) on page 112
- §5.1.6 [Skipping the Mif2Go Export and Finished dialogs](#) on page 112
- §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113
- §5.1.8 [Reordering text flows](#) on page 113
- §5.1.9 [Converting system variables to text](#) on page 114
- §5.1.10 [Preserving Word-generated cross-reference markers](#) on page 114

See also:

- §4.4 [Understanding the rules for configuration settings](#) on page 102

5.1.1 Checking output type and file extension

When you set up a conversion project (see §3.3 [Creating a Mif2Go conversion project](#) on page 78), you use the **Mif2Go Choose Project** dialog to specify the output type. **Mif2Go** produces a new project configuration file for you that contains settings for the output type you selected.

[Table 5-1](#) shows the name of the project configuration file for each output type, and the preset extension for output files. You can change the setting for the output file extension, for all output types except DITA and WinHelp.

For example, to specify a different file extension for HTML output:

```
[Setup]
; FileSuffix = suffix used for output file extension and in
; cross references
FileSuffix = .html
```

The leading dot on the extension is optional.

Table 5-1 Output types, file extensions, project configuration files

Output category	Output type	Preset output file extension	Project configuration file	Ref.
HTML-based Help	Eclipse Help	.htm	_m2eclipse.ini	12
	Microsoft HTML Help	.htm	_m2htmlhelp.ini	9
	JavaHelp	.htm	_m2javahelp.ini	11
	OmniHelp	.htm	_m2omnihelp.ini	10
	Oracle Help for Java	.htm	_m2oraclehelp.ini	11
HTML	Standard HTML	.htm	_m2html.ini	13
	XHTML	.xhtml	_m2xhtml.ini	13
XML	DITA XML	.dita (<i>not settable</i>)	_m2dita.ini	15
	Docbook XML	.ent	_m2docbook.ini	17
	Generic XML	.xml	_m2xml.ini	14
RTF	WinHelp	.rtf (<i>not settable</i>)	_m2winhelp.ini	8
	Print RTF	.rtf	_m2rtf.ini	6
Intermediate	ASCII DCL	.dcl	_m2dcl.ini	38
	FrameMaker MIF	.mif	_m2mif.ini	38

You can also use the *Export* dialog to change the file extension, just before running a conversion (see §3.7 [Choosing final conversion options](#) on page 83). If you do so, **Mif2Go** writes the new extension to the configuration file.

5.1.2 Excluding files from book conversions

To exclude FrameMaker-generated files, do one of the following:

- Uncheck the appropriate options at set-up time; see §3.4.4 [Including FrameMaker-generated files](#) on page 81.
- Use a configuration setting; see §5.5 [Converting FrameMaker-generated files](#) on page 124.

We suggest you exclude conditioned-out chapters entirely, instead of trying to hide them with conversion settings, for the following reasons:

- If temporary files from a previous conversion in which the text was *not* conditioned out are still present in the project directory, those files will be included when the

output is assembled, possibly producing unwanted results such as TOC entries for the missing chapter. Since the chapter does not produce any output, the old temporary files are not overwritten.

- Processing an essentially empty file wastes time.

5.1.3 Reusing or discarding MIF files

FrameMaker keeps document files in “Maker” format, typically with an .fm extension. This binary file format is a closely held trade secret of Adobe Systems Incorporated. FrameMaker files must be saved as MIF (Maker Interchange Format) before **Mif2Go** can read them. **Mif2Go** converts the MIF files to binary DCL (Document Coding Language) files, and finally from DCL to the output type you specify (see §1.5 [How Mif2Go works](#) on page 62).

Mif2Go ordinarily saves your FrameMaker files as MIF immediately before each conversion. However, if you have not altered a FrameMaker file since the last time it was saved as MIF, you can choose to let **Mif2Go** use the existing MIF file. You can specify this option in the **Mif2Go Export** dialog, which in turn updates the configuration file. Or, you can change this setting directly in the project configuration file:

```
[Setup]
; UseExistingMIF = No (default) or Yes (use if it exists)
UseExistingMIF = Yes
```

You can have **Mif2Go** delete old MIF files from the project directory just before starting a new conversion. To delete MIF files from the project directory before conversion:

```
[Automation]
; DeleteExistingMIF = No (default) or Yes (delete *.mif from the
; project directory before the conversion).
DeleteExistingMIF = Yes
```

When DeleteExistingMIF=Yes, **Mif2Go** deletes old MIF files from the project directory before conversion, *provided both of the following are true*:

- UseExistingMIF is *not* set, or is set to No
- **Use existing MIF file if any** is *not* checked on the *Export* dialog (see §3.7.3 [Choosing input source and disposition](#) on page 85).

5.1.4 Reusing or discarding ASCII DCL files

You can use **Mif2Go** to convert the files in your document to ASCII DCL, by choosing **ASCII DCL only** in the **Mif2Go** plug-in *Choose Project* dialog (see §3.3 [Creating a Mif2Go conversion project](#) on page 78). Because **Mif2Go** processes graphics as part of the conversion from MIF to DCL, and writes out graphics files in addition to DCL files, you can use this choice to export graphics from your document so you can alter or replace them before producing final output. Next time you run the same conversion, you can direct **Mif2Go** to use those DCL files (and your altered or replaced graphics files) instead of creating them anew:

```
[Setup]
; UseExistingDCL = No (default, make .dcb)
; or Yes (use .dcl file if it exists)
UseExistingDCL = Yes
```

When you specify UseExistingDCL=Yes, instead of creating binary DCL files (extension .dcb) and then deleting them at the end of the conversion process, **Mif2Go** uses the existing ASCII DCL files (extension .dcl), and leaves them in place.

When you are finished with a set of DCL files, to clear them out before you begin a new conversion from your document:

```
[Setup]
; DeleteExistingDCL = No (default) or Yes (delete *.dcl from the
; project directory before conversion if UseExistingDCL is not set.
DeleteExistingDCL = Yes
```

When DeleteExistingDCL=Yes, **Mif2Go** deletes both .dcl and .dcb files from your project directory before conversion.

5.1.5 Checking for broken links in HTML or XML output

When you convert a FrameMaker book to HTML or XML, **Mif2Go** can check all the interfile cross-reference and hypertext links in your output, and provide a FrameMaker “Book Error Log” that lists any broken links.

Note: Although this feature is not available for RTF, MIF, or DCL conversions, you can still check links for those outputs by running an HTML conversion on the same book, with the same conditions shown. The same link errors should be found.

To check for broken links:

```
[Setup]
; CheckLinks = No (default) or Yes (check links after running a book)
CheckLinks = Yes
; CheckLinkLog = D:\path\to\LinkLog.fm to make copy of Book Error Log
CheckLinkLog = path\to\MyBadLinks.fm
; LinkLogAlways = Yes (default) or No (do not display Book Error Log
; if no broken links are found)
LinkLogAlways = Yes
```

When CheckLinks=Yes, after converting a book, **Mif2Go** writes a notice to the FrameMaker Console window showing how many broken links were found. If **Mif2Go** finds at least one broken link (or if LinkLogAlways=Yes), **Mif2Go** displays a Book Error Log in FrameMaker, and closes only those FrameMaker files that have no link errors. The Book Error Log contains active links to the broken links in the open files.

To save the Book Error Log, assign an absolute path and file name to CheckLinkLog. **Mif2Go** copies the Book Error Log to the specified file for safekeeping; the Book Error Log itself disappears when you exit FrameMaker.

If **Mif2Go** finds more than 1,024 link errors, only the first 100 are logged. This indicates a massive problem, for which additional information from the other 924+ errors would not help.

In a few cases you might find that an interfile link reported by **Mif2Go** as broken actually works just fine in FrameMaker. This can happen when macros are involved, or can be caused by latency issues in Windows shell operations. The remedy is to re-convert the chapter containing the link, without deleting any .ref files.

5.1.6 Skipping the Mif2Go *Export* and *Finished* dialogs

Normally you will want to have both the *Choose Project* dialog and the *Export* dialog come up when you click **File > Save Using Mif2Go...** so you can change graphics or MIF usage settings before each conversion. To proceed with conversion as soon as you select the right project, you can turn off the *Export* dialog.

To skip the *Export* dialog:

```
[Setup]
; UseInitDialog = Yes (default, display before conversion) or No
UseInitDialog = No
```

You can also eliminate the *Finished* dialog that announces completion of each **Mif2Go** conversion, and that requires you to click **OK** before you can do anything else in FrameMaker.

To skip the *Finished* dialog:

```
[Setup]
; UseDoneDialog = Yes (default, display after conversion) or No
UseDoneDialog = No
```

When `UseDoneDialog=No`, the **Status** bar at the bottom of the FrameMaker book or document file window shows the message when the conversion is complete:

```
Mif2Go project finished
```

5.1.7 Specifying how to treat cases, spaces, and wildcards

You can choose how **Mif2Go** interprets FrameMaker paragraph, character, and table format names, to match them to settings in the configuration file:

```
[Options]
; CaselessMatch = Yes (default, ignore upper/lower differences) or No
CaselessMatch = Yes
; SpacelessMatch = Yes (default, ignore embedded spaces) or No
SpacelessMatch = Yes
; WildcardMatch = Yes (default, allow ? and * in settings) or No
WildcardMatch = Yes
```

The default settings help eliminate hard-to-spot typing errors. However, you might have to change one or more of these settings if any format names in your document:

- differ only in case (such as *Body* and *body*): set `CaselessMatch=No`.
- differ only by spaces (such as *Lastbullet* and *Last bullet*): set `SpacelessMatch=No`.
- contain asterisks or question marks: set `WildcardMatch=No`. See §4.6 [Using wildcards in configuration settings](#) on page 106.

5.1.8 Reordering text flows

When your FrameMaker document contains more than one tagged text flow, **Mif2Go** normally writes the flows out in the order they are encountered. **Mif2Go** ignores empty flows and flows that appear only on Master or Reference pages. However, you can interleave flows instead, or omit certain flows, perhaps because they contain text used only for cross references.

Note: Although you can skip flows and merge flows, you cannot change the order of the text flows.

To change the treatment of selected text flows (for example):

```
[TextFlows]
; flowtags to Skip or to treat as Normal (to keep in same section)
z = Skip
A = Normal
B = Normal
```

When `flowtag=Skip`, the text flow is omitted from output.

When `flowtag=Normal`, **Mif2Go** writes out content in the order in which text appears, regardless of flow.

When you do not specify either `Skip` or `Normal` for some flows, **Mif2Go** writes them out in the order encountered.

In this example, **Mif2Go** would “merge” flows A and B, skip flow z entirely, and put any other flows into the output separately.

5.1.9 Converting system variables to text

The values of FrameMaker system variables, such as date and time, are not stored in your FrameMaker files; instead, they are read from the system when you load a document into FrameMaker. Therefore, **Mif2Go** does not find values for system variables in the MIF files. (However, user-defined variables *are* present in a usable form in MIF).

*System variables
on master pages*

For print RTF output, **Mif2Go** converts the values of system variables *that appear on master pages* to their Word field equivalents. **Mif2Go** does not convert master-page content for WinHelp or HTML output.

*System variables
on body pages*

To get system date/time and file-name variables *that appear on body pages* into your output, **Mif2Go** must convert these variables to text. Specify the following setting:

```
[Setup]
; ConvertVariables = No (default) or Yes (convert to plain text)
ConvertVariables = Yes
```

Apply this setting conservatively. It is best not to convert system variables to text in *all* files, because the process requires slow FDK operations that can have unpleasant side effects. The problems are in the FDK, not in **Mif2Go**, so there is little Omni Systems can do to fix them. However, applying `ConvertVariables=Yes` to one or a few short files—for example, to the title page of a FrameMaker book—should not hurt. See §33.1 [Using a different configuration for selected files](#) on page 919.

Note: For system variables to show up in the MIF files, **Mif2Go** must read your original FrameMaker files. If you specify **Use existing MIF** on the *Export* dialog, or in your project configuration file, system variables are not converted.

See also:

§6.6 [Converting system variables to text for RTF](#) on page 157

§13.6.2 [Converting system variables to text for HTML](#) on page 437

5.1.10 Preserving Word-generated cross-reference markers

Content imported from Word via FrameMaker native Word import arrives in FrameMaker cluttered with hundreds of extra cross-reference markers, all starting with `_Toc` followed by a long number. This is an artifact of the method Word uses to create a table of contents. Your document might contain many of these markers, because Word uses them only once, and makes a new set every time it generates a table of contents. If you leave these markers in your FrameMaker document, they appear in the **Mif2Go** output as named destinations, such as a # footnote in WinHelp, or `` in HTML.

Do not try to use these markers. By default, **Mif2Go** ignores them.

However, if the markers have been used in your FrameMaker document, you can instruct **Mif2Go** to honor them, with the following option:

```
[HelpOptions] or [HTMLOptions]
; RemoveWordTocMarkers = Yes (default, ignore markers starting _Toc)
; or No
RemoveWordTocMarkers = No
```


Cross references to these markers will then work in **Mif2Go** output; but you will also have many (perhaps thousands) of extra pointless anchors in HTML or footnotes in WinHelp. They are harmless, but they do increase file size.

In future you can avoid this problem when you import files from Word. Bring them into FrameMaker as plain text, using **Copy** in Word and **Paste Special** in FrameMaker. Then re-tag per your FrameMaker template. This method takes longer, but it eliminates numerous issues you would otherwise have forever as the result of the Word import.

5.2 Logging conversion events

Whenever you convert a document, by default **Mif2Go** records conversion events in a plain ASCII log file located in the project directory. This event log lists files opened and any error messages or warnings that are produced during conversion. At the start of the next conversion run, **Mif2Go** appends the finished event log to a history file before starting a new log.

To disable logging, or to change the name or location of the log file or history file:

```
[Logging]
; UseLog = Yes (default, log as specified in this section) or No
UseLog = No
; LogFileName = name with path (absolute, or relative to project dir)
LogFileName = _m2g_log.txt
; EditorFileName = text editor executable to display log if errors
EditorFileName = notepad.exe
; ShowLog = Yes (default, display log in text editor if errors or
; warnings) or No
ShowLog = Yes
; HistoryFileName = name with path of cumulative log history, to which
; the contents of LogFileName are appended.
HistoryFileName = _m2g_history.txt
```

The default name of the log file is `_m2g_log.txt`, and the default name of the history file is `_m2g_history.txt`. Unless you specify a different path for `LogFileName` or for `HistoryFileName`, **Mif2Go** writes the log file and history file to the project directory. If you specify a relative path, that path is relative to the project directory.

At the start of a conversion **Mif2Go** appends the contents of any existing log file (named by `LogFileName`) to the history file named by `HistoryFileName`, then deletes the contents of the old log file. Because the purpose of the log is to make diagnosing problems easier, **Mif2Go** appends log entries to the history file for successive conversions. A key diagnostic approach is to compare entries from successive conversion runs, and not necessarily just the last two runs.

When `ShowLog=Yes`, if you are running the conversion from the FrameMaker plug-in, if any warnings or errors occur, the plug-in pops up the log file in the editor named by `EditorFileName`. The default editor is `notepad.exe`. If you specify a text editor that is not on the system execution path, you must include its full path in the value for `EditorFileName`. If you are running **Mif2Go** from the command line, each **Mif2Go** DLL pops up the log file if errors or warnings are encountered.

When `UseLog=Yes`, you can specify the type and the importance (or level of severity) of events **Mif2Go** reports in the log file:

```
[Logging]
; These take severity values, 1 (greatest) to 9 (least),
; or 0 to prevent logging (except for LogInfo)
; LogErrors = 1 (default, log events that terminate a process)
```

```

LogErrors = 1
; LogWarnings = 1 (default, log problems with workarounds that might
; result in undesired output)
LogWarnings = 1
; LogQueryys = 1 (default, log possible ambiguities)
LogQueryys = 1
; LogInfo = 1 (default, log process information; 0 is ignored)
LogInfo = 1
; LogDebug = 0 (default, do not log possible programming issues)
LogDebug = 0

```

Log entry types are as follows:

```

E      Error: process terminated
W      Warning: problem with a workaround
Q      Query: possible ambiguity
I      Information only
D      Debug: possible programming issue

```

By default, **Mif2Go** logs only the most important or severe events (level 1), but not less important or less severe events (levels 2 through 9). At level 1 only the most important processing events are logged, such as the start of processing for each FrameMaker file and the identity of the software module doing the processing. Unless you specify otherwise **Mif2Go** does not log events classified as debugging issues.

Note: When UseLog=Yes, process information is always logged, even if you set LogInfo=0.

Each log entry appended to the log file includes the following information:

- timestamp (if different from the previous entry), on a line by itself
- event type (E, W, Q, I, or D)
- severity level or importance (from 1 = most severe or important to 9 = least severe or important)
- event description.

Flagging uncatalogued formats

By default, **Mif2Go** logs a warning about any format that is used in your document but not defined in the FrameMaker catalog. If you get tired of seeing warnings about uncatalogued formats:

```

[Logging]
; ShowUndefinedFormats = Yes (default) or No
ShowUndefinedFormats = No

```

Recording configuration chains

In addition to logging conversion events, you can have **Mif2Go** include in the event log all the chains of configuration files and templates referenced by your project:

```

[Logging]
; LogIniChains = No (default) or Yes, list all chains
LogIniChains = Yes

```

When LogIniChains=Yes, before listing events, **Mif2Go** shows the full path of every configuration file and template used in processing, in the order they are referenced by settings in the [Templates] section. For example:

```

I1: Ini chain for Configs:
I1:  _m2omnihelp.ini
I1:  ..\_config\m2gug_htm_document.ini
I1:  ..\_config\m2gug_document.ini
I1:  g:\omnisys\m2g\local\config\local_m2omnihelp_config.ini
I1:  g:\omnisys\m2g\system\config\m2omnihelp_config.ini

```

```

I1: g:\omnisys\m2g\local\config\local_m2help_config.ini
I1: g:\omnisys\m2g\system\config\m2help_config.ini
I1: g:\omnisys\m2g\local\config\local_m2htm_config.ini
I1: g:\omnisys\m2g\system\config\m2htm_config.ini
I1: g:\omnisys\m2g\local\config\local_m2g_config.ini
I1: g:\omnisys\m2g\system\config\m2g_config.ini
I1: g:\omnisys\common\local\config\local_omsys.ini
I1: g:\omnisys\common\system\config\omsys.ini

```

This output shows the chain of general configuration files and templates referenced from starting configuration file `_m2omnihelp.ini`, for a project to generate OmniHelp from FrameMaker.

A log-file example that shows a warning:

```

Sat Oct 04 11:38:21 2008
I1: Starting log for dwhtm, h285z10
I1: Opened file "planning.htm" for HTML
I1: File ID is z102x
W2: Xref target ag123456 not found in intro.ref

```

The W2 entry is a warning, severity level 2, about a cross-reference destination missing from the **Mif2Go**-generated reference file for chapter `intro.fm`.

5.3 Identifying files and objects

To maintain interfile and intrafile references, **Mif2Go** creates object names, link destination names, and output file names from FrameMaker ObjectIDs and **Mif2Go** FileIDs. These components originate as follows:

ObjectID: Six-digit number assigned by FrameMaker to each paragraph, table, and graphic; in MIF format, this is the `<Unique ID>` tag. Or, five-digit number assigned by FrameMaker to each cross reference; in MIF format, this is the `<XRefSrcText>` tag. See §5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118.

FileID: Two- or three-character alphanumeric code assigned by **Mif2Go** to each FrameMaker file. See §5.3.4 [Working with Mif2Go FileIDs](#) on page 119.

In this section:

§5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117

§5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118

§5.3.3 [Working with FrameMaker cross-reference IDs](#) on page 119

§5.3.4 [Working with Mif2Go FileIDs](#) on page 119

5.3.1 Understanding how Mif2Go creates identifiers

To name objects and references in your document, and to name certain output files, **Mif2Go** creates identifiers of the following form:

`[L][ff]nnnnnn[.hhh]`

where *L*, *ff*, *nnnnnn*, and *.hhh* are as follows:

<i>L</i>	<i>Link destinations only:</i> X for cross references, R for hypertext links.
<i>ff</i>	FileID (<i>required for books, optional for single-file documents</i>), assigned to each FrameMaker file in <code>mif2go.ini</code> ; see §5.3.4 Working with Mif2Go FileIDs on page 119.

nnnnnn Five-digit (for cross references) or six-digit ObjectID. For output files this is usually the ObjectID of the first item: paragraph, table, or anchored frame. See §5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118.

.hhh *Files only:* period and file extension.

Mif2Go creates identifiers for the following items:

- Cross-reference anchors, for which the FrameMaker ID includes a five-digit number (for example, Rab12345)
- Hypertext link destinations (for example, Xac254678)
- Graphics files produced with FrameMaker export filters (for example, ad3f509e.gif); see §5.7.4.1 [Naming files produced by FrameMaker export filters](#) on page 133
- Split and extract files for HTML, XML, and DITA output (for example, ae9704561.htm); see §18.4.1 [Understanding how split and extract files are named](#) on page 593
- Anchored frames for HTML (for example, aa4de33f); see §23.5 [Selecting and modifying graphics](#) on page 708.
- Tables for HTML (for example, bb123412); see §24.2 [Defining sets of tables](#) on page 728.

This kind of composite identifier is sometimes referred to as an *FDK name*.

5.3.2 Working with FrameMaker ObjectIDs

You can view the ObjectID of a paragraph, table, or graphic on the FrameMaker status bar:

Paragraph: Click the paragraph, then **Shift**-click without moving the mouse. The ObjectID shows on the status bar as the **ParaID**, a six-digit decimal integer.

Table: **Shift**-click the table. The ObjectID appears on the status line as the **TableID**, a six-digit decimal integer. (Use **Ctrl**-click to see the information FrameMaker normally shows on the status bar for **Shift**-click.)

Graphic: Select the anchored frame or named reference-page frame. The ObjectID appears on the status bar as the **FrameID**, a six-digit hexadecimal number.

ObjectIDs are not always persistent

ObjectIDs are nominally unique within a FrameMaker file. However, when you hide and then show a conditional paragraph or anchored frame, *the ObjectID always changes*. For tables, the ObjectID persists only if you do *not* include the table anchor when you hide and show the table.

If you copy an item, normally the copy gets a new ObjectID. However, if you cut an entire paragraph, FrameMaker does *not* assign a new ObjectID when you paste the paragraph; instead, FrameMaker copies the ObjectID of the original paragraph. That is, FrameMaker assigns a new ObjectID if you *copy* and paste text, but not if you *cut* and paste text.

Very infrequently, FrameMaker re-uses an existing ObjectID number when assigning a new ID. To ensure that new numbers are unique, FrameMaker increments a “next number” value, which is stored in the document file. However, FrameMaker does not check for conflicts with any existing values before assigning an ID number. If you happen to specify an object with a duplicated number as the destination of a cross-reference or hypertext link, you can get erroneous links in output from **Mif2Go**.

<i>Resolve duplicate ObjectIDs</i>	To correct the problem of duplicate ObjectIDs, you must delete, then recreate, one of the duplicate markers in FrameMaker. If you <i>cut</i> and then paste the marker, FrameMaker keeps the old number; however, if you <i>copy</i> and paste, FrameMaker assigns a new number. Select one of the paragraphs with the duplicate ObjectID, copy (not cut) the paragraph, delete the original, then paste the copy back in place. The paragraph will now have a new ObjectID.
<i>Do not delete <Unique ID> tags</i>	Because Mif2Go uses ObjectIDs based on MIF <code><Unique ID></code> and <code><XRefSrcText></code> tags, do not delete these tags from MIF files. The tags are targets for ObjectID hypertext links, such as those created by generating FrameMaker TOC and IX files. Remove the tags, and you break all links from existing generated files (indexes and lists). Although FrameMaker recreates the links next time you generate, the new links will not have the same numeric IDs. Removing MIF tags is a particularly bad idea if you use ObjectIDs to specify properties in the configuration file for graphics and tables. After updating your document you would have to look at every graphic and every table, get the new ID for each, and edit the configuration file accordingly.

5.3.3 Working with FrameMaker cross-reference IDs

Although FrameMaker tries to assign unique five-digit IDs to cross references, in a single FrameMaker document two cross-reference markers might have identical ID numbers. Cross references with duplicate ID numbers can work properly within FrameMaker, because FrameMaker actually uses a much longer ID that includes the format name and the first several words of the referenced paragraph along with the ID number. However, references to different markers with the same ID number would lead to a single destination in **Mif2Go**-generated output. This is because the FrameMaker “verbose” IDs are not valid in RTF or HTML, so **Mif2Go** trims them back to just the ID number, which *usually* is unique within a file.

To resolve isolated instances of duplicate cross-reference IDs, the best approach is to recreate one of the cross references in FrameMaker. To find duplicates, generate an Index of Markers for each FrameMaker file, including markers only of type **Cross-Ref**, and scan through the index for any ID numbers that are followed by more than one page reference. This problem is rare, but it is nasty when it bites.

If you are converting Structured FrameMaker files to HTML, also see §5.8 [Converting structured documents](#) on page 135.

5.3.4 Working with Mif2Go FileIDs

In this section:

- §5.3.4.1 [Understanding how and where FileIDs are assigned](#) on page 120
- §5.3.4.2 [Replacing FileIDs with custom identifiers](#) on page 121
- §5.3.4.3 [Updating files and references when FileIDs change](#) on page 121
- §5.3.4.4 [Keeping legacy FileIDs in the configuration file](#) on page 122

See also:

- §5.7.4.1 [Naming files produced by FrameMaker export filters](#) on page 133
- §6.11.5.1 [Identifying Word link destinations with FileIDs](#) on page 179
- §8.9.1 [Identifying WinHelp jump destinations with FileIDs](#) on page 273
- §18.4.1 [Understanding how split and extract files are named](#) on page 593
- §C.4 [Renaming or relocating the Mif2Go FileID file](#) on page 1027

5.3.4.1 Understanding how and where FileIDs are assigned

By default, **Mif2Go** uses FileIDs in the construction of cross references and hypertext links. When you set up a **Mif2Go** project from within FrameMaker, **Mif2Go** assigns a two-letter FileID to each FrameMaker file in your document, and stores those FileIDs in file `mif2go.ini`, which by default resides in the same directory as your FrameMaker document.

There should be just one copy of `mif2go.ini` for a given FrameMaker book or document. However, if you are converting multiple books that reference each other, they must all use the same copy of `mif2go.ini`; see §2.7 [Setting up multiple interlinked HTML projects](#) on page 75 and §19.6.4 [Enabling links to files in other projects](#) on page 623.

FileIDs are not required for a one-file document

If your FrameMaker document is a single file, you can direct **Mif2Go** *not* to use FileIDs:

```
[WordOptions] or [HelpOptions] or [HTMLOptions]
; UseFileIDs = Yes (default, needed for identifying xrefs) or No
UseFileIDs = No
```

When `UseFileIDs=No`, links that **Mif2Go** makes from cross references and generated hypertext links do not include any identification of the file(s) involved.

FileIDs are needed for multiple files

Unless the output from your conversion project will be *only* a single file, let **Mif2Go** use FileIDs, so links do not get confused between files if a cross-reference number or ObjectID is not unique. This is especially important for graphics, where ObjectIDs (converted to hexadecimal) are used to name the graphics files. Without FileIDs, a graphic produced for one file could easily overwrite a graphic made for another file, resulting in an incorrect display.

Using FileIDs prevents problems with identical ObjectIDs occurring in two different FrameMaker files (common for files originating from the same template), but FileIDs do not help with duplicate IDs within any one file.

FileIDs in mif2go.ini

When `UseFileIDs=Yes`, by default **Mif2Go** stores FileID assignments in a FileID file that **Mif2Go** creates, named `mif2go.ini`, located in the source directory with your FrameMaker document. You can specify a different name and location for the FileID file:

```
[Setup]
; IDFileName = name of file that contains FileIDs for this project
IDFileName = D:/path/to/mif2go.ini
```

Specify an absolute path for `IDFileName`, because this reference has to work from unpredictable locations. See §C.4 [Renaming or relocating the Mif2Go FileID file](#) on page 1027.

FileIDs in old configuration files

If you are using an old configuration file that contains FileID assignments, and you want **Mif2Go** to use the assignments in your configuration file instead of assignments in `mif2go.ini`, specify the following option:

```
[Setup]
; UseLocalFileID = No (default, use IDFile IDs)
; or Yes (use [FileIDs] here)
UseLocalFileID = Yes
```

See §5.3.4.4 [Keeping legacy FileIDs in the configuration file](#) on page 122.

FileID assignments

A FileID assignment takes the form `fmfile=id`, where:

`fmfile` is the name (without extension) of a FrameMaker file in your document
`id` is a two- or three-letter identifier.

FileID assignments are listed in section `[FileIDs]` of the FileID file (or possibly in section `[FileIDs]` of an old configuration file). For example:

```
[FileIDs]
Intro = aa
Chapter1 = ab
Chapter2 = ac
```

Normally, the only time a new FileID assignment is needed is after you add a new chapter to a book. If **Mif2Go** makes such an assignment during set-up, **Mif2Go** uses the value of `[FDK]NextFileID` listed in `mif2go.ini`.

5.3.4.2 Replacing FileIDs with custom identifiers

Mif2Go-generated FileIDs are unique for all **Mif2Go** projects listed in `.prj` files that reside in the same directory as `mif2go.ini`. If you want more associative names for FileIDs, after you have set up a conversion you can edit `mif2go.ini` to replace generated FileIDs with identifiers of your own choosing. These identifiers can consist of any valid file-name characters. Stick to lowercase letters if possible, keep the identifiers short, and make sure they are unique.

Note: Until your system is working flawlessly, do not change the FileIDs **Mif2Go** writes to `mif2go.ini`.

Keep in mind these restrictions:

- FileIDs must be unique for all current FrameMaker files in your document.
- FileIDs must be short (two or three alphanumeric characters).
- FileIDs may not contain spaces.
- A FileID should not end in a digit.

For example:

```
[FileIDs]
Intro = in
Main = ma
Summary = su
```

Watch out for case sensitivity. Unless you specify `[Options]CaselessMatch=No`, **Mif2Go** regards AA, Aa, aA, and aa as identical; see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113.

Duplicate FileIDs are allowed only for replaced FrameMaker files

There should be no duplicate FileID assignments in `mif2go.ini`, with one exception. If you change the name of a FrameMaker file in your document, and there are cross references to that file from other files, you might want to map both the old and new FrameMaker file names to the same FileID, so that existing references work. For example:

```
[FileIDs]
Intro = in
Main = ma
NewMain = ma
Summary = su
```

5.3.4.3 Updating files and references when FileIDs change

The current method of naming FileIDs (and interfile links) was introduced in **Mif2Go** Version 3.2. If you update a file created using an older version of **Mif2Go**, you might have to update all files that it references, and that reference the updated file. Otherwise you might encounter broken links, because the internal link identifiers might be using different FileIDs.

Note: Whenever FileIDs change, you must update settings in any configuration sections that reference the previous FileIDs, such as the [Graph*] sections.

5.3.4.4 Keeping legacy FileIDs in the configuration file

If you are still using a **Mif2Go** configuration created before the introduction of `mif2go.ini`, you might have FileIDs in the main configuration file:

```
[FileIDs]
; original filename (no ext) = prefix ID for text and graphic objects.
fmfile = id
```

To continue using FileIDs listed in `mif2htm.ini` or `mif2rtf.ini`, set the following option:

```
[Setup]
; UseLocalFileID = No (default, use mif2go.ini IDs)
; or Yes (use [FileIDs] here)
UseLocalFileID = Yes
```

When `UseLocalFileID=Yes`, **Mif2Go** uses the FileIDs (if any) listed in your project configuration file. If a FrameMaker file (either the current file or a referenced file) is not already listed in configuration section [FileIDs], **Mif2Go** tries to create a unique FileID by using the last three characters of the base file name (such as `er2` for `Chapter2.mif`). If that sequence is already in use, **Mif2Go** uses the last four characters, and so on, until **Mif2Go** finds an unused sequence or is using the entire base file name. FileIDs created this way are *not* added to the [FileIDs] section of your configuration file.

When `UseLocalFileID=No` (the default), **Mif2Go** uses the FileIDs in `mif2go.ini` instead; this is the preferred method.

Note: Do not add a [FileIDs] section to the main configuration file for new projects; use `mif2go.ini` instead.

5.4 Applying FrameMaker conditions and variables

To produce multiple outputs from a single FrameMaker source, you might need several different combinations of FrameMaker variables and condition **Show/Hide** settings. Usually, we advise establishing those combinations in FrameMaker conversion templates, one for each output type; see §30.7 [Applying FrameMaker conversion templates](#) on page 863.

However, if you need so many FrameMaker templates that maintenance of common formats across all variations would be difficult, you can specify values for FrameMaker user variables and condition **Show/Hide** settings in your **Mif2Go** project configuration file instead.

It is best not to use these settings to condition out entire chapters of a book; instead, create a separate FrameMaker book that does not include the unwanted chapters. See §5.1.2 [Excluding files from book conversions](#) on page 110.

In this section:

§5.4.1 [Applying condition Show/Hide settings](#) on page 123

§5.4.2 [Replacing values of FrameMaker user variables](#) on page 123

5.4.1 Applying condition Show/Hide settings

Note: The method described in this section might not work in FrameMaker version 10, in which case you will need to use the older method, importing FrameMaker templates; see §2.4 [Importing formats from a conversion template](#) on page 67.

To apply FrameMaker condition **Show/Hide** settings to **Mif2Go** output:

```
[Setup]
; SetFrameConditions = No (default) or Yes (set per [ConditionsShown]
; after template import, if any).
SetFrameConditions = Yes
```

To specify which conditions to show and which to hide:

```
[ConditionsShown]
; Condition name = Yes to show or No to hide. Any not mentioned
; are left in their current state; any not found are ignored.
; Note: not for use in a configuration template.
WantedCondition = Yes
UnwantedCondition = No
```

If you are importing a FrameMaker conversion template, these conditions are applied after import, so that any conditions you do *not* list retain the settings they had in the imported template.

For example, to include condition *HelpOnly* and exclude condition *PrintOnly*:

```
[Setup]
SetFrameConditions = Yes

[ConditionsShown]
HelpOnly = Yes
PrintOnly = No
```

If any conditions are actually altered, **Mif2Go** closes the FrameMaker file without saving, same as after template import. If the file was open before processing, **Mif2Go** reopens the original file.

If you are producing HTML or DITA XML output, you can express FrameMaker conditions as element attribute settings. See:

§13.10 [Converting conditions to HTML attributes](#) on page 446

§15.12 [Converting conditions to DITA attributes](#) on page 533.

5.4.2 Replacing values of FrameMaker user variables

To replace FrameMaker user variables with **Mif2Go** macro variables:

```
[Macros]
; ReplaceFrameVars = No (default) or Yes (replace Frame variables with
; the correspondingly named macro variable, if any)
ReplaceFrameVars = Yes
```

For example, to supply run-time values for FrameMaker user variables *ProductName* and *ProductVersion*:

```
[MacroVariables]
ProductName = Ace Gadget
ProductVersion = 10.3
```

Any FrameMaker user variables not listed in [MacroVariables] retain the values you gave them in FrameMaker.

See also:

§28.3.2 [Assigning values to macro variables](#) on page 797

[§28.3.5 Treating FrameMaker user variables as macro variables](#) on page 801

5.5 Converting FrameMaker-generated files

In this section:

[§5.5.1 Converting FrameMaker TOC and IX files](#) on page 124

[§5.5.2 Preventing conversion of other generated files](#) on page 125

[§5.5.3 Activating hypertext links in a converted index](#) on page 125

[§5.5.4 Making See and See also index entries into useful links](#) on page 125

5.5.1 Converting FrameMaker TOC and IX files

When you are converting a FrameMaker book, you can choose whether to include FrameMaker-generated contents and index files in the process:

```
[Setup]
; UseFrameTOC = Yes (default, except for Help formats, DocBook,
; and DITA), or No (default for Help formats, DocBook, and DITA)
UseFrameTOC=Yes
; UseFrameIX = Yes (default, except for Help formats, DocBook,
; and DITA), or No (default for Help formats, DocBook, and DITA)
UseFrameIX=Yes
```

The default value is the same for both UseFrameTOC and UseFrameIX, and depends on the output type you specify:

<u>Output type</u>	<u>Default</u>
Print RTF (Word, WordPerfect)	Yes
WinHelp	No
Standard HTML	Yes
DITA	No
DocBook	No
EclipseHelp	No
JavaHelp	No
MS HTML Help	No
OmniHelp	No
Oracle Help for Java	No
XHTML	Yes
XML	Yes
MIF only	Yes
ASCII DCL only	Yes

Note: These are default values in the sense that **Mif2Go** presets them that way in the *SetUp* dialog. But if the settings are not present in your project configuration file because someone deleted them, the default for both is Yes for all output types.

If you are creating a Help system, you should not need either TOC or IX; by default, **Mif2Go** generates contents and index for Help systems. See [§7.3.2 Including FrameMaker TOC and IX in Help systems](#) on page 205.

To make sure TOC entries that contain character formatting become active links in their entirety, see [§5.10.2 Making an entire paragraph into a hotspot](#) on page 138.

5.5.2 Preventing conversion of other generated files

By default, **Mif2Go** automatically converts generated files other than TOC and IX. To prevent **Mif2Go** from converting other generated files:

```
[Setup]
; UseFrameGenFiles = Yes (default for all formats except DITA)
; or No (default for DITA)
UseFrameGenFiles=No
```

To make sure entries that contain character formatting in LOF, LOM, and other “list of” generated files become active links in their entirety, see §5.10.2 [Making an entire paragraph into a hotspot](#) on page 138.

5.5.3 Activating hypertext links in a converted index

When you convert a FrameMaker index, only the page numbers in an entry become hypertext links, because each index entry can have multiple links to different parts of your document.

To activate the index links, you must apply a character format to the index-entry page numbers, in FrameMaker. Without the character format, multiple page references for an index entry might cause a crash.

To apply a character format to index page numbers:

1. In the FrameMaker IX file, define a new character format; for example, *IXpgnum*. The character format can be set to “As Is”.
2. On the IX Reference page of the IX file, find the line with paragraph format *IndexIX*, which contains the `<$pagenum>` element, and apply character format *IXpgnum* to the entire line.
3. Save the IX file, and generate the book. In the Body pages of the regenerated IX file you will see that the page numbers are still links but the index-entry text is not, just as before; so locked FrameMaker documents and PDF files are unaffected.

For HTML output, you can replace each page number with a clickable image; see §13.8.1.3 [Replacing page numbers with symbols or images](#) on page 442.

See also §5.5.4 [Making See and See also index entries into useful links](#) on page 125.

5.5.4 Making See and See also index entries into useful links

A FrameMaker-generated index includes, for every `<$nopage>` entry, a cross reference to the original marker location in your document. Such references are not very useful. If your index includes a lot of *See* and *See also* entries, consider using IndexRef, a FrameMaker plug-in available from Sundorne Communications:

<http://www.sundorne.com/FrameMaker/IndexRef/indexref.htm>

IndexRef changes `<$nopage>` *See* and *See also* entries in a FrameMaker IX file so the links point to the referenced index entries instead of to the original markers in the text. When you convert a FrameMaker index processed by IndexRef, **Mif2Go** preserves the corrected *See* and *See also* links in the output (except for Help systems; see §7.5 [Configuring index entries for Help systems](#) on page 211).

Note: For OmniHelp output, **Mif2Go** redirects `<$nopage>` links, whether or not you use IndexRef; see §10.7.6 [Redirecting See and See also index entries](#) on page 359.

5.6 Generating/updating before converting

If you need to regenerate a table of contents, index, or other generated file because an imported template would change page references for these files, or if you do *not* update cross-references and other links for an entire book before converting, you can have **Mif2Go** generate/update your book before proceeding with conversion (and after applying a conversion template, if you specified one). Specify the following setting:

```
[Setup]
; GenerateBook = No (default) or Yes (generate after import)
GenerateBook=Yes
```

Generate/update during conversion is an expensive choice in terms of time and memory, because all book files must be open at the same time. Best to generate/update in FrameMaker before you start the conversion, if at all possible.

Note: See §30.7.4 [Avoiding template-related disasters](#) on page 866 for important information about what *not* to do if **Mif2Go** encounters a problem while converting your document.

5.7 Processing graphics

This section provides a brief overview of options and methods for getting the graphics in your FrameMaker document into an appropriate format.

If some graphics referenced by your document are in a format that is not appropriate for the output type you select (see §31.1 [Choosing an appropriate graphics format](#) on page 869), you have the following options:

- Omit the graphics from the output.
- Recreate the graphics in a different format.
- Convert the graphics to a different format.

In this section:

§5.7.1 [Understanding which graphics are included](#) on page 126

§5.7.2 [Choosing how to convert graphics](#) on page 127

§5.7.3 [Choosing when to convert graphics](#) on page 131

§5.7.4 [Identifying exported graphics files](#) on page 133

See also:

§3.7.4 [Figuring out graphics export options](#) on page 85

§31 [Working with graphics](#) on page 869

5.7.1 Understanding which graphics are included

Mif2Go processes graphics that occur in the following places in a FrameMaker document, unless you specify otherwise:

- anchored frames on body pages
- unanchored frames on body pages (*HTML output only*)
- unanchored frames and images on master pages (*RTF output only*)
- named frames on reference pages.

For HTML output, by default **Mif2Go** attaches any unanchored frame on a body page to the first paragraph on that page; see §23.5.5 [Eliminating graphics in unanchored frames](#) on page 713.

For RTF output, by default **Mif2Go** attaches any unanchored frame on a master page to a header or footer.

See also:

§31.2.5.7 [Converting graphics on reference pages](#) on page 885

§31.2.5.9 [Converting unanchored graphics on body pages](#) on page 886

5.7.2 Choosing how to convert graphics

You have two (for HTML/XML) or three (for RTF) ways to convert graphics to another format. **Mif2Go** handles the first two of the following:

§5.7.2.1 [Using Mif2Go native graphics processing](#) on page 128 (RTF output only)

§5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129 (RTF or HTML output)

§5.7.2.3 [Using third-party graphics converters](#) on page 130 (RTF or HTML output)

[Table 5-2](#) and [Table 5-3](#) summarize the types of processing your graphics are likely to need, based on:

- the document output type (HTML/XML or RTF)
- the original graphic format (JPEG, BMP, etc.) and access type (referenced or embedded)
- whether an anchored frame contains a single image with no added elements (such as callouts), or contains multiple images or added elements.

This is just a starting point; you might need to experiment with different methods and different settings to find the best way to handle graphics for your **Mif2Go** project.

For example, if you are converting graphics for use in WinHelp, be sure you understand when to use WMF and when to use BMP; see §8.6.1 [Choosing a graphics format for WinHelp](#) on page 263.

See §31 [Working with graphics](#) on page 869 for more information.

Table 5-2 Basic graphic conversion options for HTML/XML

Image format	Access type	How to convert for use in HTML/XML output	Ref.
GIF, JPEG, PNG**	Referenced	OK as is; no conversion needed, except possibly rescaling	23.9
	Embedded	Export from FrameMaker (<i>default</i>)	31.2.3
FM vector, FM equation, or compound* illustration		Use FM export filters to convert to GIF, JPEG, or PNG	5.7.2.2
Any other format**	Referenced	Use FM export filters to convert to GIF, JPEG, or PNG, <i>or</i>	5.7.2.2
		Use a 3rd-party tool to convert to GIF, JPEG, or PNG	5.7.2.3
	Embedded	Use FM export filters to convert to GIF, JPEG, or PNG, <i>or</i> :	5.7.2.2
		1. Export from FrameMaker, <i>then</i> 2. Use a 3rd-party tool to convert to GIF, JPEG, or PNG	5.7.3.2 5.7.2.3

*Multiple images in the same anchored frame, or image with callouts, etc.

**Single image, alone in its anchored frame

Table 5-3 Basic graphic conversion options for RTF

Image format	Access type	How to convert for use in RTF output	Ref.
BMP or WMF		OK as is; Mif2Go embeds BMP images in WMF	5.7.2.1
FrameImage, FM vector		Mif2Go converts automatically to WMF	5.7.2.1
FM equation or compound* illustration		Use FM export filters to convert to WMF or BMP	5.7.2.2
EPS**	Referenced	Use a 3rd-party tool (or FM filters) to convert to BMP or WMF	5.7.2.3
	Embedded	1. Set <code>EpsiUsage=EPS</code> or <code>EpsiUsage=Retain</code> (<i>print RTF only</i>), export from FrameMaker; <i>then</i> 2. Use a 3rd-party tool to convert to BMP or WMF	31.2.2.3 31.2.3
Any other format**	Referenced	Use a 3rd-party tool (or FM filters) to convert to BMP or WMF	5.7.2.3
	Embedded	1. Export from FrameMaker, <i>then</i> 2. Use a 3rd-party tool to convert to BMP or WMF	5.7.3.2 5.7.2.3

*Multiple non-BMP, non-WMF images in the same frame, or non-BMP, non-WMF image with callouts.
**Single image, alone in its anchored frame

See also:

- §3.7.4 [Figuring out graphics export options](#) on page 85
- §6.14 [Managing graphics for print RTF](#) on page 186
- §8.6 [Managing graphics for WinHelp](#) on page 263
- §23 [Including graphics in HTML](#) on page 703

5.7.2.1 Using Mif2Go native graphics processing

RTF output only

Mif2Go can convert several graphic types to RTF-friendly formats. **Mif2Go** has full built-in support for converting the following formats to BMP or WMF:

- BMP (Windows bitmap, .bmp)
- WMF (Windows metafile, .wmf)
- OLE objects
- FrameImage (Sun raster, .rf)
- FrameMaker vector graphics (images created with FrameMaker drawing tools)

Mif2Go processes graphics of these types to produce WMF output, by default. You get the same result when you do one of the following:

- In the **Mif2Go Export** dialog, choose **Write graphics for equations**
- In the RTF configuration file, set the following options:

```
[Graphics]
UseGraphicPreviews = No

[Setup]
WriteEquations = Yes
WriteAllGraphics = No
```

The built-in **Mif2Go** graphics processing does a better job than the FrameMaker export filters for many graphics, and has much finer controls for adjusting things like callout sizes. Also see §6.14.5 [Managing callouts added to graphics](#) on page 190.

Not for tables in anchored frames

Mif2Go native graphics processing fails miserably on multi-cell tables inside anchored frames. For such cases, the FrameMaker export filters do a better job.

5.7.2.2 Using FrameMaker graphic export filters

Mif2Go can use FrameMaker graphic export filters to produce graphics files from the contents of anchored frames in your document. Converting graphics this way can take awhile, because the FrameMaker filters are not known for speed. This is the only way **Mif2Go** can export equations to external files.

Note: This built-in conversion is provided as a convenience only; we do not consider it to be acceptable for quality document production.

In this section:

§5.7.2.2.1 [Understanding when to use FrameMaker export filters](#) on page 129

§5.7.2.2.2 [Understanding FrameMaker filter limitations](#) on page 129

§5.7.2.2.3 [Specifying options for FrameMaker export filters](#) on page 130

§5.7.2.2.4 [Specifying graphic output format and DPI](#) on page 130

For a complete list of FrameMaker export filter settings, see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883.

5.7.2.2.1 Understanding when to use FrameMaker export filters

You might need to resort to FrameMaker export filters for your graphics in the following situations, depending on the type of output format:

[RTF output](#) (Word, WinHelp)

[HTML output](#) (HTML, HTML-based Help, XHTML, XML).

RTF output Use FrameMaker graphic export filters for RTF output if your document contains either of the following:

- Multi-cell tables that are inside anchored frames.
- One or more referenced graphics for which both of the following are true:
 - The graphics are in formats *other than*:
 - › BMP (“DIB” in the FrameMaker import dialog),
 - › WMF (most of which work; but see §5.7.2.2.2 [Understanding FrameMaker filter limitations](#) on page 129), or
 - › OLE (which works only if **Mif2Go** can dig out the embedded WMF preview image, no small feat).
 - You have not mapped the graphics to equivalent BMP or WMF files (created outside of **Mif2Go**), and you do not want them used in their current format.

HTML output Use FrameMaker graphic export filters for HTML output only if your document contains one or more referenced graphics for which both of the following are true:

- The graphics are either (or both) of the following:
 - in formats *other than* JPEG, GIF, or PNG
 - not alone in their anchored frames (for example, they include FrameMaker drawing elements such as arrows or callouts).
- You have not mapped the graphics to equivalent JPEG or GIF files (created outside of **Mif2Go**), and you do not want them used in their current format.

5.7.2.2.2 Understanding FrameMaker filter limitations

FrameMaker export filters can respond poorly to slightly corrupt imported graphics that FrameMaker itself tolerates. The export filter might crash and cause a **Mif2Go** error; this is probably the case if the FrameMaker status bar shows “Writing Graphics” at the time of the crash.

You might *not* want to use FrameMaker graphics export filters in any of the following circumstances:

- You need better resolution than screen, and your original imported graphics have significantly better resolution.
- The FrameMaker export filters put out incorrect images (WMF), or images with artifacts (BMP on Windows NT).
- Your graphics are in EPS format; the FrameMaker filters use only the preview image, and do a poor job with it.
- Your shrinkwrapped graphics will be viewed on a colored background in the output, and the white padding is undesirable.

5.7.2.2.3 Specifying options for FrameMaker export filters

To have **Mif2Go** use FrameMaker graphic export filters:

```
[Graphics]
; UseGraphicPreviews = No (default)
; or Yes (use preview bitmaps for frames)
UseGraphicPreviews = Yes

[Setup]
; WriteAllGraphics = No (default)
; or Yes (write all anchored frames as graphics files)
WriteAllGraphics = Yes
```

5.7.2.2.4 Specifying graphic output format and DPI

To specify the output format for graphics produced with FrameMaker export filters:

```
[Setup]
GraphicExportFormat = format
```

where *format* is one of the following:

BMP, TIFF, WMF, JPEG, PNG, EPS, PICT, CGM, GIF, IGES

The default format depends on the output type:

```
RTF output:      GraphicExportFormat = BMP
HTML output:     GraphicExportFormat = JPEG
```

You can set the DPI for the FrameMaker export filter to use. However, specifying DPI does not change the resolution of the image; all it does is scale the image to another size. Deviations from 96 DPI (especially small deviations) can make graphic text unreadable. You might have to experiment with different DPI settings to get the best graphic quality.

To specify DPI for graphics produced with FrameMaker export filters:

```
[Setup]
; GraphicExportDPI = number (from 50 to 1200, default 96)
GraphicExportDPI = 96
```

For HTML output, you can override the `GraphicExportDPI` value with a different DPI value; see §23.9 [Scaling images for HTML](#) on page 719.

5.7.2.3 Using third-party graphics converters

Several graphics programs, such as the following, can convert images from one format to another:

Graphic Workshop Pro	http://www.mindworkshop.com
Adobe Illustrator	http://www.adobe.com/products/illustrator/main.html
Corel Paint Shop Pro	http://www.jasc.com/ (redirect)

GhostScript/GhostView <http://www.cs.wisc.edu/~ghost/> (for EPS graphics)

You can use third-party tools only on graphics that exist as separate external files; you cannot use them directly on embedded graphics or on FrameMaker vector graphics. However, you can run **Mif2Go** to export embedded graphics, then use third-party tools to convert them:

1. Run **Mif2Go** to export embedded graphics as separate files; see §5.7.3.2 [Processing embedded graphics separately](#) on page 132.
2. Use a third-party graphics program to alter or replace the graphics; save each using the *same file name* as the original (referenced) or exported (embedded) graphic, but with a different extension.
3. If you specified ASCII DCL as the output type when you exported graphics, to avoid rewriting graphics and recreating DCL files, do one of the following:

- In the **Mif2Go Export** dialog, check **Use Existing DCL file, if any**.
- In the configuration file, set the following option:

```
[Setup]
UseExistingDCL = Yes
```

4. In the configuration file, set the following options:

```
[Graphics]
UseGraphicPreviews = No

[Setup]
WriteEquations = Yes
WriteAllGraphics = No
```

5. **RTF output.** Indicate that you want file names to be mapped, and specify both old and new file extensions (without leading periods):

```
[Graphics]
FileNames = Map

[GraphFiles]
old_extension = new_extension
```

6. **HTML output.** Indicate that you want to keep the original file names, and specify the new extension:

```
[Graphics]
UseOriginalGraphicNames = Yes
GraphSuffix = new_extension
```

7. Run **Mif2Go** again.
8. Remember to uncheck **Use Existing DCL file, if any** (or set `UseExistingDCL=No`) for later runs, if you change any text in your FrameMaker document.

See also §31.3 [Replacing and relocating graphics files](#) on page 887.

5.7.3 Choosing when to convert graphics

Former versions of **Mif2Go** processed all graphics in a FrameMaker document before carrying out the rest of the conversion. The current version of **Mif2Go** first identifies which graphics actually need attention; then, after the rest of a file is converted, processes only those graphics. For example, a reference-page graphic is produced only if it is actually used on a body page, and if its use in the output is not suppressed by some other configuration setting.

In this section:

§5.7.3.1 [Processing all graphics first](#) on page 132

[§5.7.3.2 Processing embedded graphics separately](#) on page 132

5.7.3.1 Processing all graphics first

In either of the following circumstances you might want **Mif2Go** to process graphics before processing the rest of your document:

- Your workflow relies on an older **Mif2Go** method for RTF output that processed all graphics first.
- You want *all* graphics written out to files, whether or not the graphics are used in your document or will be included in the output.

To have **Mif2Go** process graphics before converting the rest of the document:

```
[Setup]
; GraphicsFirst = No (default, write only needed graphics, after DCL
; process), or Yes (old way, write graphics of the types specified,
; before DCL process)
GraphicsFirst = Yes
```

When GraphicsFirst=Yes, if you also direct **Mif2Go** to use FrameMaker export filters, **Mif2Go** writes out to files *all* graphics in anchored frames on body pages, and *all* graphics in frames on reference pages, whether or not they are used in your document. This process can take quite awhile, and use a staggering amount of disk space. Sometimes reference and master pages contain a lot of unused graphics, which **Mif2Go** normally avoids converting.

See also:

[§5.7.3.2 Processing embedded graphics separately](#) on page 132

[§38.4.3 Exporting embedded graphics via ASCII DCL output](#) on page 1012

5.7.3.2 Processing embedded graphics separately

You can create external graphics files from images that were originally copied into a FrameMaker document, without converting the document itself, when you specify ASCII DCL as the output type. If you have copied-in graphics, and need to modify them, this method can get the graphics out of your document with minimum fuss, saving most of the time it would take to run a full conversion.

Image quality is retained; size and added elements are not

Because **Mif2Go** uses native graphics export for this purpose, the full original quality of each graphic, whatever it was during import, is retained for each image. However:

- Images are not scaled to the size displayed in FrameMaker, so they might be larger or smaller than you expect.
- The exported graphics do not include any transformations or elements added in FrameMaker after import.

Elements added in FrameMaker are not exported

This method *does not work* to export compound graphics: embedded or referenced images with elements added in FrameMaker, using FrameMaker drawing tools. All you get is the original imported image, in its original size and shape, without whatever was added in FrameMaker.

Steps to export embedded graphics

To export embedded graphics to external files:

1. Open the FrameMaker document.
2. From the FrameMaker **File** menu, choose **Set Up Mif2Go Export...**
3. Create a project to output ASCII DCL; see [§38.2.2 Setting up a FrameMaker MIF project](#) on page 1006.
4. Specify the following settings in your project configuration file:

```
[Setup]
GraphicsFirst = Yes

[GraphExport]
ImportGraphics = Export
```

5. Save your project configuration file.
6. From the FrameMaker **File** menu, choose **Save Using Mif2Go...**
7. In the **Mif2Go Export** dialog:
 - 7.1. Choose **Write graphics for equations**.
 - 7.2. *Do not* check **Write only graphics, no text**.
 - 7.3. Click **OK**.

*All graphics
become files*

Mif2Go exports to individual files all graphics that were copied into your FrameMaker document, including OLE objects. **Mif2Go** extracts a WMF from each OLE object and exports the WMF as an individual file; see §31.2.4 [Exporting images and creating files from OLE objects](#) on page 881.

*Exported files get
arbitrary names*

Each new file is named with the first few letters of the FrameMaker file name, followed by a number, and with the correct extension for the type of graphic. You can specify how many letters and digits to use in the graphics file names; see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134.

Once you have the graphics in separate files, you can use a third-party graphics program to batch-convert them to an appropriate format for the type of output you want **Mif2Go** to generate. See §31.3 [Replacing and relocating graphics files](#) on page 887.

See also:

- §3.7.4 [Figuring out graphics export options](#) on page 85
- §5.7.3.1 [Processing all graphics first](#) on page 132
- §38.4.3 [Exporting embedded graphics via ASCII DCL output](#) on page 1012
- §31.2.3 [Exporting and converting embedded graphics](#) on page 877.

5.7.4 Identifying exported graphics files

Mif2Go creates a name for each graphic file generated from your FrameMaker document. How the file name is constructed depends on the origin of the graphic and the method used to create the file.

In this section:

- §5.7.4.1 [Naming files produced by FrameMaker export filters](#) on page 133
- §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134
- §5.7.4.3 [Naming external graphic metafiles](#) on page 134

5.7.4.1 Naming files produced by FrameMaker export filters

Mif2Go can create graphics files via FrameMaker export filters; see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883 for more information.

The graphic file name consists of the FileID (if you have chosen to use FileIDs), followed by the FrameMaker ObjectID of the graphic. You can specify whether to use FileIDs and how many ObjectID digits to include:

```
[Setup]
; UseGraphicFileID = Yes (default) or No (single-file projects only)
UseGraphicFileID = Yes
```

```
; GraphicNameDigits = 6 (default), range 4 to 8
GraphicNameDigits = 6
```

See §5.3 [Identifying files and objects](#) on page 117 for information about FileIDs and ObjectIDs.

5.7.4.2 Naming files produced from embedded graphics

Mif2Go can create files from graphics that were copied into FrameMaker, to make those graphics accessible to other converters, such as Graphic Workshop (see §31.2.3 [Exporting and converting embedded graphics](#) on page 877). Because FrameMaker does not retain the original names of embedded graphics, **Mif2Go** must create new names for the resulting graphics files. Each appearance of a graphic produces a new file, even if the same graphic has appeared before.

By default, **Mif2Go** uses the first four characters of the FrameMaker file name, and adds a four-digit number. You can specify different numbers of characters and digits for names of exported graphics files:

```
[GraphExport]
; ExportNameChars = chars from base file name
;   to use in export file names
ExportNameChars = 4
; ExportNumDigits = number of digits to use in export file names
ExportNumDigits = 4
```

Make sure file names are unique

If you have more than 9,999 instances of graphics that were copied into a FrameMaker file, you must increase the number of digits used. If you are trying to keep to 8.3 file names, you might need to reduce the number of letters accordingly.

If your conversion project includes FrameMaker files whose names do not differ in the first four characters, you must increase the number of characters to use. For example, if your FrameMaker files are named Chapter1.fm, Chapter2.fm, and so on, you must use all eight letters to avoid name conflicts between graphics from different files. In that case you will not be able to keep to 8.3 names.

Better to import by reference

Better to take the newly exported graphics files, give them proper names, and go back into FrameMaker and import them *by reference* in place of the original embedded images, being careful to select the image itself for replacement and not the anchored frame; then run the conversion again.

Why not to embed graphics

Why not embed graphics? Besides making your FrameMaker files very large, embedded graphics can be irretrievably lost when you save a FrameMaker file that contains them. The problem is the size available for the Windows TEMP directory, which by default is located on the disk drive that tends to have the least available space, usually the C drive. While saving a document file FrameMaker writes out an expanded version of every embedded graphic to a temporary file in the TEMP directory, then reads them all back in while writing the .fm file. If you run out of room on the drive containing the TEMP directory, FrameMaker merrily continues saving without complaint, but is no longer able to write the graphics back into the saved file.

5.7.4.3 Naming external graphic metafiles

Mif2Go generates external metafiles for WinHelp; see §31.2.6 [Embedding bitmap graphics in WMF for WinHelp](#) on page 886 for more information.

By default, **Mif2Go** uses the first five characters of the base file name, and adds a three-digit number. You can change the number of characters and digits:

```
[Graphics]
; MetaNameChars = chars from base file name
```

```

; to use in external WMF file names
MetaNameChars = 5
; MetaNumDigits = number of digits
; to use in external WMF file names
MetaNumDigits = 3

```

Each appearance of a graphic produces a new file, even if the same graphic has appeared before; and for WinHelp each subscript, superscript, and symbol also produces a metafile. To decide how many characters and digits to specify, consider all of the following:

- If a FrameMaker file you are converting to WinHelp contains more than 999 images, you must increase the number of digits used.
- If you are trying to keep to 8.3 file names, you might have to reduce the number of letters accordingly.
- If your conversion project includes FrameMaker files whose names do not differ in the first three characters, you must increase the number of characters to use.

5.8 Converting structured documents

Mif2Go can convert Structured FrameMaker documents. However, the conversion is based on paragraph and character formats, not on elements, so you must use distinct format names to get distinct effects in the output. Specify a different format in your FrameMaker EDD for each element type that needs to be visually distinct in the output, rather than using *Body* with overrides for everything. In other words, use the EDD to create exactly the sort of formatting used in unstructured files.

*Overrides require
font tags for
HTML output*

If you are converting to HTML and you are stuck with a one-format EDD, you can retain in the HTML most (but not all) of the overrides; however, you have to accomplish this with font tags and align attributes, which makes CSS pretty much useless for the resulting HTML.

*Cross references
rely on element
attributes*

Mif2Go supports cross references in Structured FrameMaker documents by relying on attributes of the referenced element, instead of on paragraphs and markers. To convert cross references in Structured FrameMaker files, by default **Mif2Go** uses the value of element attribute `Id` as the target for cross references, and recognizes elements with attribute `Idref` as references to element IDs. If your Structured FrameMaker files use attributes with other names for these purposes, you must tell **Mif2Go** what names to look for; otherwise the cross references **Mif2Go** generates will not work as expected.

To specify the names of cross-reference and element ID attributes in your Structured FrameMaker document:

```

[HTMLOptions] OR [WordOptions]
; IDAttrName = name of structured-element ID attribute, default "Id"
IDAttrName = Id
; IDRefAttrName = name of structured-element cross-reference
; attribute, default "Idref"
IDRefAttrName = Idref

```

By default, **Mif2Go** does not use structure tags and attributes for any other purpose.

*Map attributes to
markers*

To capture the values of other Structured FrameMaker attributes, you can map the attributes to FrameMaker markers. For example, if the `linkref` attributes in your Structured FrameMaker document contain names you want to use for HTML split files:

```

[AttributeMarkers]
; Structured FrameMaker attribute name = FrameMaker marker name
linkref = FileName

```

See §34.8.3 [Using custom markers to name output files](#) on page 947.

See also:

§29 [Working with FrameMaker markers](#) on page 831

5.9 Converting equations

In this section:

§5.9.1 [Understanding how equations are processed](#) on page 136

§5.9.2 [Specifying equation size and DPI](#) on page 136

§5.9.3 [Specifying equation output format](#) on page 137

§5.9.4 [Providing a file-name suffix for equations](#) on page 137

§5.9.5 [Positioning equations in RTF output](#) on page 137

5.9.1 Understanding how equations are processed

Mif2Go produces graphics for FrameMaker native equations, essentially screen renderings in WMF or bitmap form. The content is not editable as such, but the rendition appears correctly in all output types.

For DITA XML output, equations become <image> elements in a <fig> element; see §15.7.5 [Including MathFullForm equations in <alt> elements](#) on page 518. For DCL representation, **Mif2Go** keeps both the native FrameMaker MathFullForm and the graphic rendering. Although all current output types except DITA use only the graphic, the MathFullForm is available also, in DCL.

Mif2Go always uses FrameMaker export filters to write native FrameMaker math equations as graphics, whether or not you direct **Mif2Go** to use those filters for graphic images in your document.

Despite the unfortunate key name, the following setting does not actually affect the production of *equations*; provisionally, it affects the production of *graphics*:

```
[Setup]
; WriteEquations = No (default) or Yes (write equations as graphics)
```

When WriteEquations=No, equations (*and* graphics, provided [Graphics]UseGraphicPreviews=Yes) are converted with FrameMaker export filters.

When WriteEquations=Yes, *only* equations are converted with FrameMaker export filters. Unless [Graphics]UseGraphicPreviews=Yes, the graphics in your document are processed some other way, or not at all.

The purpose of WriteEquations=Yes is to allow you to experiment with settings for equations without processing the rest of the document. Usually you would not change this setting in the configuration file. Instead you would use the **Mif2Go Export** dialog; see §5.9.2 [Specifying equation size and DPI](#) on page 136.

See also:

§3.7.4 [Figuring out graphics export options](#) on page 85

5.9.2 Specifying equation size and DPI

Native FrameMaker export uses 72 DPI as a default for equations. This is much too small, and produces unreadable text for all equation sizes except the largest. The best DPI value varies; bigger is not always better. Generally, a value between 100 and 150 yields acceptable results. Equations need bigger values than other graphics.

To adjust the size and resolution of equations:

```
[Setup]
; EquationExportDPI = number (from 50 to 1200, default 120)
EquationExportDPI = 120
; EquationFrameExpand = percentage of original size (default 125)
EquationFrameExpand = 125
```

For best results, set `EquationFrameExpand` to a value about 4% greater than the value for `EquationExportDPI`.

To experiment with different values without converting the whole document over and over again, in the **Mif2Go Export** dialog choose the following:

- **Write graphics for equations**
- **Write only graphics, no text.**

See §3.7.4 [Figuring out graphics export options](#) on page 85.

5.9.3 Specifying equation output format

For RTF output, the default format for equations is WMF, which is a vector format that generally gives the best rendition. However, if equations do not look right in WMF, try BMP instead. No other choice would give better results. BMP might result in poorer print rendition of equation text, but visual rendition should be nearly as good as WMF:

```
[Setup]
; GraphicExportFormat = BMP, TIFF, WMF (RTF default),
; JPEG (HTML default), GIF, PNG, EPS, PICT, CGM, or IGES
GraphicExportFormat = BMP
```

For HTML output, the default format for equations is JPEG.

See §31.2.5.5 [Specifying graphic output format and DPI](#) on page 884.

5.9.4 Providing a file-name suffix for equations

If you need to isolate equation image files from other generated graphics for postprocessing, you can specify a file-name suffix for equations, to be inserted before the file extension:

```
[WordOptions] or [HelpOptions] or [HTMLOptions]
; EqSuffix = suffix followed by period followed by file extension
EqSuffix = eq.jpg
```

For example, with this setting a generated equation file name would look like:

```
aa1234567.eq.jpg
```

When you specify a value for `EqSuffix` you must include the file extension, which overrides the extension implied by the setting for `GraphicExportFormat`; see §5.9.3 [Specifying equation output format](#) on page 137.

5.9.5 Positioning equations in RTF output

By default, for RTF output **Mif2Go** embeds equations in Windows Metafiles (WMFs), so in-line equations can be positioned correctly in WinHelp. This is not essential for Word RTF, or for equations that are not in line, because other methods are available for alignment. However, you can adjust equations horizontally only if they are in WMFs.

To adjust the scale and position of equations in RTF output:

```
[WordOptions] or [HelpOptions]
; EmbedEqsInWMFs = Yes (default, scale to size using WMFs) or No
```

```

EmbedEqsInWMFs = Yes
; EqVertAdjust = half-points to adjust equations down
; (negative for up)
EqVertAdjust = 8
; EqHorAdjust = half-points to adjust equations right
; (negative for left)
EqHorAdjust = 2

```

These equation settings are not available for HTML output.

5.10 Creating hotspots for hypertext links

When you insert a FrameMaker hypertext marker in a paragraph to create a link, you can create a visible hotspot for the link.

Note: A hotspot can contain *only one* marker. If you need different hypertext markers for different output types, duplicate the entire hotspot, and use conditional text to hide all but one at a time of the duplicated hotspots.

In this section:

§5.10.1 [Delimiting a hotspot with a character format](#) on page 138

§5.10.2 [Making an entire paragraph into a hotspot](#) on page 138

§5.10.3 [Delimiting a hotspot with a color](#) on page 139

See also:

§7.8.2 [Defining a pop-up hotspot](#) on page 226

§8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274

§19.5.2 [Converting FrameMaker hypertext links to HTML](#) on page 619

5.10.1 Delimiting a hotspot with a character format

You can create a hotspot for a hypertext link by applying a character format to just those words you want highlighted and activated. The span of the hotspot (the active area of the link) is determined by the span of the character format in which you place the marker.

Sometimes by accident a marker is placed just before, or just after, character-formatted text; **Mif2Go** watches for this situation, and treats such markers as though they were inside the formatted area. However, if the marker is not in or adjacent to the formatted text, the unformatted part of the paragraph where the marker is located becomes the hotspot, which is practically never what you want. Also, if character formats have been applied to any part of a paragraph, the link extends both ways from the marker, until the character format changes.

5.10.2 Making an entire paragraph into a hotspot

When you insert a marker in a paragraph to which no character formats have been applied, the entire paragraph becomes a link.

If a paragraph includes character formatting, but you want the entire paragraph to be the hotspot, assign property `ParaLink` to the paragraph format:

```

[HTMLParaStyles] or [HelpStyles] or [WordStyles]
; paragraph format = ParaLink
; ParaLink prevents any char formats in the named para format from
; affecting the hotspot area for a link in that para.

```


This is especially handy for entries in converted FrameMaker-generated files. In generated files, entries that include a format change have their associated hypertext links truncated at the point of change. For example, in the TOC for your document you might have a definition like this:

```
<SomeFormat><$paranum> <Default Para Font><$paratext> <$pagenum>
```

The active portion of the link would be only the paragraph number. To make the entire text of all TOC entries work as hypertext links:

```
[HTMLParaStyles] or [HelpStyles] or [WordStyles]
*TOC = ParaLink
```

5.10.3 Delimiting a hotspot with a color

To make colored text a hotspot when only the text color changes, and the color is applied with an override instead of a character format, set the following option:

```
[HTMLOptions] or [HelpOptions]
; UseHyperColor = No (default) or Yes (treat any non-black as hyper)
UseHyperColor = Yes
```

5.11 Repurposing FrameMaker markers

You can reuse the content of most FrameMaker markers, and also make new custom marker types, by remapping a marker type to one or more other marker types. See §29 [Working with FrameMaker markers](#) on page 831 for more information.

The standard FrameMaker marker types are as follows:

Author
Comment
Conditional Text (*cannot be cloned or redefined*)
Cross-Ref
Equation
Glossary
Header/Footer \$1
Header/Footer \$2
HTML Macro
Hypertext
Index
Subject

Any marker name *not* listed here is the name of a custom marker type.

*Many custom
marker types
have predefined
effects*

Because many marker types have a dedicated purpose or require specific content, you must be careful about remapping to custom marker types. [Table 29-1](#) on page 832 lists the custom marker types predefined for **Mif2Go** conversions. For example, if you expect to convert your FrameMaker document to any HTML output type, do not try to remap to marker types that are predefined for WAI support; see §34.1.2 [Using markers to add links and instructions](#) on page 935.

To remap a marker type to one or more other marker types:

```
[Markers]
; marker type name = one or more marker type names
FM_Marker = OtherMarker AnotherMarker ...
```

The original marker type is no longer in effect after remapping, unless you remap it to itself.

You must observe the following restrictions:

- The **Conditional Text** marker type cannot be remapped.
- Names of marker types you are remapping *to* (names to the right of the = sign) may not contain spaces or commas (those to the left of the = may contain spaces and commas).

You can remap any marker type (except **Conditional Text**) to:

- one or more existing or predefined marker types
- any new marker type(s) you name to the right of the = sign.

For example, to add all FrameMaker **Subject** markers to the index, and also clone them as **ALink** markers for their help topics (custom marker type **ALink** is predefined by **Mif2Go**):

```
[Markers]
Subject = Index ALink
```

You can also remap most of the **Hypertext** subtypes; see §29.3 [Remapping marker types and hypertext commands](#) on page 836.

6 Converting to print RTF

This section shows you how to specify options for converting to Microsoft Word, or to WordPerfect; most settings apply to both. The RTF produced for Word can also be viewed in OpenOffice and StarOffice. Topics include:

- §6.1 [Converting to Word: a one-way street](#) on page 141
- §6.2 [Setting up a print RTF project](#) on page 145
- §6.3 [Adjusting output for different versions of Word](#) on page 149
- §6.4 [Converting a FrameMaker book to print RTF](#) on page 150
- §6.5 [Specifying document layout options](#) on page 151
- §6.6 [Converting system variables to text for RTF](#) on page 157
- §6.7 [Converting paragraph and character formats](#) on page 158
- §6.8 [Converting tabs and spaces](#) on page 163
- §6.9 [Specifying font usage](#) on page 166
- §6.10 [Modifying text appearance](#) on page 170
- §6.11 [Converting cross references and hypertext links](#) on page 174
- §6.12 [Converting generated files to print RTF](#) on page 181
- §6.13 [Converting tables to print RTF](#) on page 184
- §6.14 [Managing graphics for print RTF](#) on page 186
- §6.15 [Including RTF code for Word output](#) on page 194
- §6.16 [Turning on revision tracking in Word](#) on page 194
- §6.17 [Managing Word output after conversion](#) on page 195
- §6.18 [Converting to OpenOffice or StarOffice](#) on page 197

See also:

- §2.2 [Naming FrameMaker formats](#) on page 66, for usages to avoid in your FrameMaker document.

If you are creating WinHelp, see:

- §7 [Producing on-line Help](#) on page 199
- §8 [Generating WinHelp](#) on page 243

You must use a separate project directory and separate configuration files for WinHelp; Word and WinHelp RTF files are not compatible.

6.1 Converting to Word: a one-way street

The intended purpose of **Mif2Go** conversion from FrameMaker to print RTF is to support review of FrameMaker-maintained documents by reviewers who use only Word. Converting **Mif2Go**-generated RTF files back to FrameMaker is not a viable option. When **Mif2Go** converts a FrameMaker document to Word, you lose a lot of FrameMaker features; some are converted to plain text, and others are merely simulated in Word.

Word output can be challenging, especially if you must support multiple versions of Word. Microsoft changes the semantics of RTF control words regularly, without documenting these changes.

In this section:

- §6.1.1 [Understanding differences in implementation](#) on page 142

- §6.1.2 [Understanding differences in file sizes](#) on page 143
- §6.1.3 [Understanding why round-tripping is not an option](#) on page 143
- §6.1.4 [Migrating a document from FrameMaker to Word](#) on page 144
- §6.1.5 [Developing a workflow using Word for reviews](#) on page 144

6.1.1 Understanding differences in implementation

Word is not FrameMaker. Although **Mif2Go** does a good job of emulating FrameMaker features that do not exist in Word, there are limits. You can easily create in FrameMaker constructs that are beyond the capabilities of Word.

When **Mif2Go** converts a document to Word, differences in implementation between FrameMaker and Word can cause loss of formatting, or require workarounds for some features. These features include:

- [Autonumbers](#)
- [Format names](#)
- [Page layouts](#)
- [Rotated text or tables](#)
- [Sidehead formats](#)
- [Text in anchored frames](#)
- [Frame Above/Below](#)
- [Inter-paragraph spacing](#)
- [Headers and footers](#)
- [Equations](#)
- [Change bars](#)

<i>Autonumbers</i>	FrameMaker and Word implement autonumbering quite differently. By default, autonumbers become plain text in Word. For review purposes, what is important is the accurate <i>text</i> depiction of autonumbers in Word. When Mif2Go renders autonumbers as plain text, the numbers stay fixed, even if a reviewer interpolates a new list item. You do not want the numbers changing in a review situation; it makes the real changes harder to find among the artifacts. However, Mif2Go can generate live autonumbers in Word, as SEQ fields; see §6.7.5 Converting autonumbered formats on page 160.
<i>Format names</i>	Some FrameMaker formats are renamed by Word itself, not by Mif2Go . For example, if you have paragraph formats <i>Heading</i> and <i>heading</i> in your FrameMaker document, Word calls the second format <i>heading1</i> , because in this respect Word is not case sensitive. Also, if you have a character format with the same name as a paragraph format, Word treats them the same; in Word, it is just one namespace.
<i>Page layouts</i>	Many common FrameMaker page layout features, such as headers that extend down next to the body text, do not work in Word. You might have to redesign master pages, or replace master pages temporarily by importing a conversion template. See §6.5.1 Understanding page layout restrictions on page 151.
<i>Rotated text or tables</i>	Word does not support rotated text or rotated tables. Word does not even support rotated table cells; but see §6.13 Converting tables to print RTF on page 184 for a workaround.
<i>Sidehead formats</i>	Sideheads go into RTF text boxes. This works well when the text box is created in RTF; but try creating another like it in Word, or even try adjusting its position in Word, and you get a real mess. If you want to be able to add new sideheads in Word, you have to convert existing sideheads from FrameMaker as plain left-aligned heads. If your sideheads are in text frames inside anchored frames, each such text frame has to be alone in its anchored frame.

<i>Text in anchored frames</i>	By default, anchored frames are rendered as graphics in Word. You can tell Mif2Go to render anchored frames as text <i>when possible</i> ; see §31.5.6.3 Converting graphic text to text on page 902. Although this works when there is a single text frame in the anchored frame, it does not work when the anchored frame contains multiple text frames or both text and graphics. Word cannot nest text boxes.
<i>Frame Above/Below</i>	Word does not support Frame Above or Frame Below paragraph properties. To emulate these properties in Word, for each such frame Mif2Go inserts a small metafile in the RTF output; see §6.7.7 Converting reference frames for Word on page 162 for a way to eliminate the frames instead.
<i>Inter-paragraph spacing</i>	Word and FrameMaker compute the vertical space between paragraphs differently; it is not always possible to achieve an exact match. See §6.10.2 Adjusting paragraph spacing on page 170.
<i>Headers and footers</i>	Converting FrameMaker master-page content to Word header/footer content is the most problematic part of a Word conversion. See §6.5.1 Understanding page layout restrictions on page 151 and §6.5.9 Converting headers and footers on page 154.
<i>Equations</i>	The equation engine in FrameMaker actually models the mathematics; equations in Word are for display only. WMF images created by FrameMaker (and by Mif2Go) for equations produce sharp images. Mif2Go uses FrameMaker graphic export filters to make an image (WMF, for Word) of the frame containing the equation, then embeds that WMF in the Word RTF at a size based on the FrameMaker original, scaled as you specify. By default, Mif2Go enlarges equations a bit to improve readability; you can specify exactly how much. See §5.9 Converting equations on page 136.
<i>Change bars</i>	Unlike FrameMaker, Word does not include change bars as a text property. Instead, Word uses Revision Tracking, which is a far more complex way of identifying changes. Revision Tracking requires information that FrameMaker does not include. Therefore, Mif2Go support for converting FrameMaker change bars is limited to giving <i>Deleted</i> text the strike-through property in Word. To make this work, you would have to Show (rather than Hide) the <i>Deleted</i> conditional text in FrameMaker, possibly by using a conversion template; see §2.4 Importing formats from a conversion template on page 67. Mif2Go does not identify <i>Inserted</i> text; you would have to use a character format.

6.1.2 Understanding differences in file sizes

The RTF files **Mif2Go** produces can be quite large if the document you are converting contains bitmap graphics. You can minimize the size of RTF files by using 256-color bitmaps (instead of 24-bit true color), but even 256-color bitmaps do not compress well in RTF. However, once an RTF file is loaded in Word, Word can use internal compression methods to store the images more efficiently; then, when you save the file as `.doc`, the size will be smaller.

For files containing only text, the opposite holds: an RTF file is smaller than the `.doc` version of the same file.

6.1.3 Understanding why round-tripping is not an option

Mif2Go developers have conducted design studies to determine what it would take to round-trip between FrameMaker and Word. Their conclusion: round-tripping a document of any complexity from FrameMaker to Word and back is not feasible. Too much is lost. The time you “save” by bringing a Word document into FrameMaker is greatly exceeded by the time it takes to rebuild what you need in FrameMaker from the resulting rubble.

<i>Use Word for review</i>	If your purpose is review, use Mif2Go to produce Word files with <i>Track Changes</i> locked on (see §6.16 Turning on revision tracking in Word on page 194), then examine the changes in Word and edit the FrameMaker file by hand. Do you really want reviewers to add writing mistakes and style blunders that you might not notice? If you must copy/paste anything, use one of the plain-text add-ons, or paste into Notepad and recopy before pasting into FrameMaker. (If you paste RTF directly into FrameMaker you will pay (and pay, and pay) forever after, because Word artifacts cause issues in FrameMaker you cannot fathom.)
<i>Migrate from Word to FrameMaker</i>	If you need to go through this process only once, to get a legacy Word document into FrameMaker, use plain text and tag it all from scratch. Recreate indexing, hypertext links, cross references, and tables, and import graphics by reference. It might take a while, but the result will be a nice stable FrameMaker document. Do this for every review cycle? No one would even try.

6.1.4 Migrating a document from FrameMaker to Word

If you plan to maintain your document in Word instead of in FrameMaker, the transition is not simple. Numerous FrameMaker features that **Mif2Go** emulates in Word, such as sideheads, are not maintainable in Word; see §6.1.1 [Understanding differences in implementation](#) on page 142.

Mif2Go can convert numbering to an active form for Word, but not to the numbering Word normally uses. Instead **Mif2Go** creates SEQ fields; see §6.7.5 [Converting autonumbered formats](#) on page 160. This is because Word normal numbering is notorious for failing in large files, requiring hours of work to recreate. Although the SEQ fields are stable, you must copy/paste them into new paragraphs that you create.

TOC generation in Word is entirely different from TOC generation in FrameMaker. A Word TOC depends on fixed style names *Heading1* through *Heading9*. You would have to map the FrameMaker paragraph formats that you want included in the TOC to one of those heading styles. And of course Word knows nothing about FrameMaker options for formatting a TOC.

Migrating from FrameMaker to Word has a very high price. Make sure you are willing to pay (and pay, and pay) before you go down that hard road.

6.1.5 Developing a workflow using Word for reviews

If you make style changes in Word after conversion, you have to do this task over and over, each time you revise the document. Instead, make all changes in FrameMaker itself, before conversion, and use a conversion template to change styles; see §2.4 [Importing formats from a conversion template](#) on page 67. **Mif2Go** imports the template during conversion, without altering your original FrameMaker files; only the MIF used by **Mif2Go** is affected by the import.

*Do not try to maintain in Word any documents **Mif2Go** converts from FrameMaker.* The conversion is fast and easy; do your ongoing document work in FrameMaker, and produce Word copies on demand for reviewers, following these guidelines:

- Make sure reviewers turn on revision tracking in Word; or turn it on for them (see §6.16 [Turning on revision tracking in Word](#) on page 194).
- Make sure reviewers turn off **View > Hidden Text** in Word. **Mif2Go** uses Word hidden text to emulate some FrameMaker features; see §6.10.8 [Hiding content in Word](#) on page 173.
- When you get an edited file back, open it in Word and look at the changed areas.

- To bring new material into your FrameMaker document, copy and paste as Plain Text.
- Handle all formatting in FrameMaker.
- Import any new graphics by reference from their original graphics files, not from the images in Word.

Writers should evaluate each Word revision, then incorporate acceptable changes into FrameMaker using plain-text (*not* rich-text) copy/paste. This does not take as long as you might think, and it is a very good idea for writers to review changes carefully. Editing changes back into the FrameMaker document ensures that this important step is carried out with the authorial attention it needs.

See also:

§34.6 [Supporting document review in Word](#) on page 943

6.2 Setting up a print RTF project

When you set up a print RTF project from within FrameMaker, if configuration file `_m2rtf.ini` is not already present in the project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

To add or change any of the options described in this section, edit configuration file `_m2rtf.ini`, located in the project directory.

In this section:

- §6.2.1 [Creating a print RTF project](#) on page 145
- §6.2.2 [Choosing set-up options for a print RTF project](#) on page 146
- §6.2.3 [Specifying output file extension](#) on page 147
- §6.2.4 [Specifying the default output language and code page](#) on page 147
- §6.2.5 [Constraining the number of bookmarks in Word](#) on page 148
- §6.2.6 [Importing a Word template](#) on page 148

6.2.1 Creating a print RTF project

To create a print RTF project:

1. Create a project directory for RTF files, separate from the directory where your FrameMaker document is. DITA files are located.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens.
3. Name your print RTF project, and browse to the project directory you created in [Step 1](#) (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
4. Select one of the following output types:
 - Word 7/95 Print RTF
 - Word 8/97 Print RTF
 - WordPerfect Print RTF
5. Choose options in the *Set Up Print RTF Project* dialog (see §6.2.2 [Choosing set-up options for a print RTF project](#) on page 146).
6. Dismiss the *Conversion Designer* dialog.
7. Use a text editor to refine your choices in the resulting `_m2rtf.ini` configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).

6.2.2 Choosing set-up options for a print RTF project

When you select Word or WordPerfect as the output type for a new project, the *Set Up* dialog shown in [Figure 6-1](#) opens. [Table 6-1](#) shows the corresponding settings in the configuration file.

See also:

§3.4 [Choosing project set-up options](#) on page 79

Figure 6-1 Set Up Print RTF Project

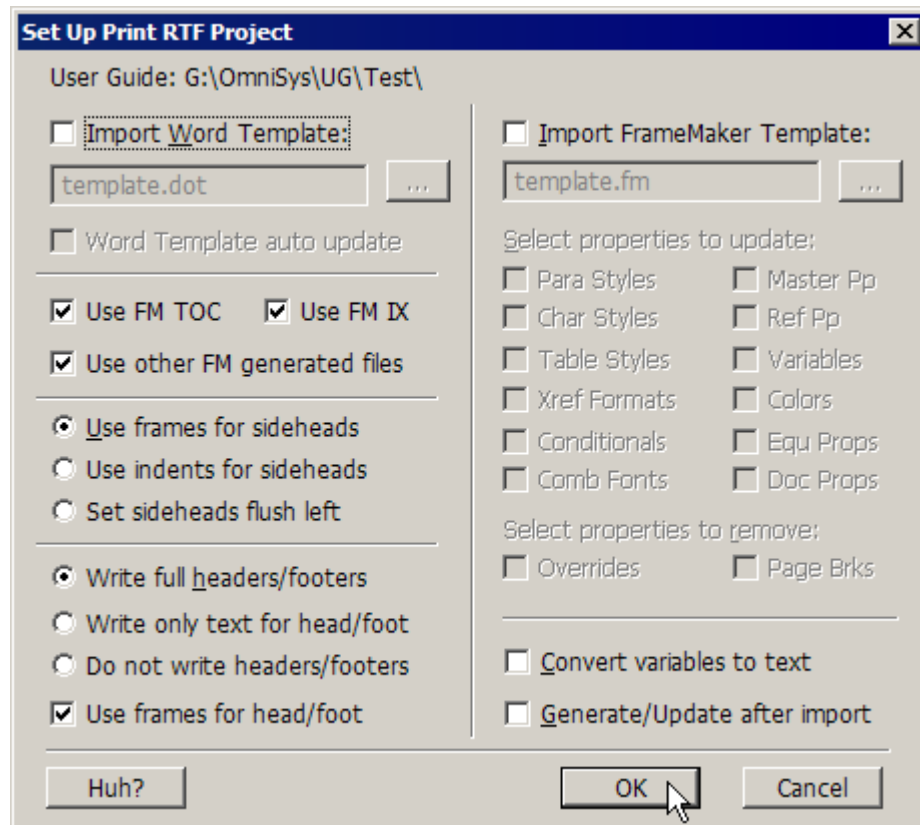


Table 6-1 Print RTF set-up options and configuration settings

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Import Word Template	[WordOptions]	Template= <i>filename</i> .dot	No default	6.2.6
Path to template file	[WordOptions]	Template= <i>filename</i> .dot	No default	6.2.6
Word Template auto update	[WordOptions]	TemplateAutoUpdate=Yes	Yes	6.2.6
Use frames for sideheads	[WordOptions]	Sideheads=Frame	No default	6.7.3
Use indents for sideheads	[WordOptions]	Sideheads=Indent	No default	6.7.3
Set sideheads flush left	[WordOptions]	Sideheads=Left	No default	6.7.3
Write full headers/footers	[WordOptions]	HeadFoot=Standard	Standard	6.5.9.5
Write only text for head/foot	[WordOptions]	HeadFoot=Text	Standard	6.5.9.5
Do not write headers/footers	[WordOptions]	HeadFoot=None	Standard	6.5.9.5
Use frames for head/foot	[WordOptions]	HFframed=Yes	Yes	6.5.9.5

6.2.3 Specifying output file extension

By default, **Mif2Go** produces RTF files with extension `.rtf`.

To specify a different file extension for RTF files:

```
[Setup]
FileSuffix=.ext
```

All versions of Word support RTF input. To produce `.doc` or `.docx` files, the RTF files **Mif2Go** produces must be loaded in Word and then saved as the desired output; see §6.17.1 [Supporting more than one version of Word](#) on page 195.

If you are converting to WordPerfect, also specify:

```
[WordOptions]
; WordPerfect = No (default) or Yes to override all features
; WP does not tolerate
WordPerfect = Yes
```

6.2.4 Specifying the default output language and code page

If you plan to produce Word output in a language other than US English, you can specify the following for several languages:

[Language or locale identifier](#)

[Code page](#).

*Language or
locale identifier*

To specify a language or locale identifier for print RTF output:

```
[Defaults]
; Language is the decimal Unicode language, or hexadecimal locale
; identifier, for the RTF default language, overriding the type in
; the source doc if given.
Language = 0x409
```

The default language is US English (`Language=1033` or `Language=0x409`). If you specify a value for `Language`, that value overrides any language specification in your FrameMaker document.

You can use the following decimal Unicode values:

US English	1033 (default)
UK English	2057
Oz English	3081
German	1031

Mif2Go supports the following hexadecimal locales (always include the `0x`):

US English	0x409 (default)
Greek	0x408
Russian	0x419
Turkish	0x41F
Czech (for CE)	0x405
Japanese	0x411
Traditional. Chinese	0x404
Simple Chinese	0x804
Korean	0x412

Code page To specify the Windows ANSI code page to use:

```
[Defaults]
; CodePage is the Windows ANSI code page number
CodePage = 1252
```

The value of CodePage is the Windows ANSI code page number, one of the following:

English	1252 (default)
Greek	1253
Russian	1251
Turkish	1254
Czech	1250
Japanese	932
Traditional Chinese	950
Simple Chinese	936
Korean	949

To specify whether to include a space after Unicode characters:

```
[Defaults]
; SpaceAfterUnicode = No (default, good for Cyrillic and Greek),
; or Yes (best for Asian languages)
SpaceAfterUnicode=No
```

6.2.5 Constraining the number of bookmarks in Word

By default, **Mif2Go** includes bookmarks in Word for all references between documents, and for all index ranges. However, Word has a limit of 16,379 bookmarks per document. If you are converting a very large document with many references, you might need to reduce the number of bookmarks.

To omit bookmarks in Word for index ranges:

```
[WordOptions]
BookmarkIXRanges = No
```

To omit bookmarks in Word for interfile references;

```
[WordOptions]
ExternalXrefs = No
```

When ExternalXrefs=No, if you are converting a multi-chapter FrameMaker book, cross references between chapters will no longer update. However, jumps between chapters will still work. See §6.11.5.2 [Creating Word bookmarks for interfile cross references](#) on page 179.

6.2.6 Importing a Word template

Importing a Word template is a last-resort procedure. It is much safer to modify formatting as needed by importing a FrameMaker template (see §2.4 [Importing formats from a conversion template](#) on page 67), so the output from **Mif2Go** is already the way you want it to look. The RTF **Mif2Go** produces is correct and valid, but what Word does with it during template import is way beyond our control.

Although a Word template updates only style definitions, you can use a single entry in a Word template to alter the properties of all FrameMaker formats merged to the same RTF style. However, when it comes to affecting Word through its own methods, what **Mif2Go** can do is limited. If you can produce the Word styles you need by using a FrameMaker conversion template instead of a Word template, that is a much safer approach. Some

FrameMaker features, such as sideheads, simply cannot be handled by a Word template. See §2.4 [Importing formats from a conversion template](#) on page 67.

If your Word template has headers and footers, for example, they will not appear in the output, because Word headers and footers work differently from FrameMaker headers and footers. Word template headers and footers are used when you create new documents in Word, but not when you apply the template to an existing Word document. If you want headers and footers in the Word output from **Mif2Go**, you must supply them in a FrameMaker template instead. Do whatever you must in FrameMaker to get them to come out the way you want them to look in Word, regardless of how awful they look in FrameMaker. See §6.5.9 [Converting headers and footers](#) on page 154.

To import a Word template.

```
[WordOptions]
; Template = name or full path of template file to attach
Template=MyWordTemplate.dot
; TemplateAutoUpdate = Yes (default, use template style defs) or No
TemplateAutoUpdate=Yes
```

Unless both options are set, the properties of the original style remain *as is* in the text.

If setting `TemplateAutoUpdate=Yes` does not apply the styles, after converting the document you might have to choose **Tools > Templates and Add-Ins...** in Word, and check **Automatically update document styles**.

6.3 Adjusting output for different versions of Word

Microsoft makes significant changes to the underpinnings of every new version of Word. RTF output from **Mif2Go** that looks fine in one version of Word might not look quite right in another version. One of the most highly visible differences is the scaling of images. If the graphics in your FrameMaker document look much too large or much too small in Word, see §6.14.7 [Preserving graphics scale in Word](#) on page 191.

Consider which version(s) of Word will be used to view your **Mif2Go** print RTF output:

[Word 2003 \(the default\)](#)
[Word 7/95 and earlier versions](#)
[Word 8/97 and later versions](#)
[Multiple versions of Word](#)

Word 2003 (the default)

By default, **Mif2Go** produces RTF output tuned for Word 2003. If you are running **Mif2Go** from within FrameMaker, this is what you get when you choose **Word 8/97** in the *Choose Project* dialog (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).

Word 7/95 and earlier versions

If your RTF output files will be viewed in a version of Word *earlier* than Word 8/97, set the following option:

```
[WordOptions]
; Word8 = Yes (default, for Word8/Office97 and later) or No (earlier)
Word8 = No
```

[Table 6-2](#) lists the main differences in the RTF code that **Mif2Go** generates, based on the value specified for `Word8`.

Table 6-2 RTF differences between Word 7/95 and later versions

Feature	Word 7/95	Word 8/97 and later versions	Ref.
Hypertext fields	Not produced	Produced by default	6.11
Table straddles	Column only	Row and column	6.13
Table cell vertical alignment	No	Yes	6.13
Graphics scale units	twips	himetric (Word 8, 9, and 10 only)	6.14.7
Image field name	IMPORT	INCLUDEPICTURE	6.14.2.3

Word 8/97 and later versions

If your RTF output files will be viewed in a version of Word *later* than Word 8/97, use the default value (Word8=Yes), and consider which of the following additional options you might need:

```
[WordOptions]
; Word2000 = No (default) or Yes (for Word 9/2000)
; Word2002 = No (default) or Yes (for Word 10/XP)
; Word2003 = Yes (default) for Word 11/2003 or No
; Word2007 = No (default) or Yes (for Word 12/2007)
; Word2009 = No (default) or Yes (for Word 13/2009)
; Word2010 = No (default, or Yes (for Word 14/2010)
```

In practice, word2003 through word2010 produce the same output; if any one of these is set to Yes, the result is the same. However, specifying either Word2000=Yes or Word2002=Yes (or both) and *not* any of Word2003 through Word2010 will produce graphics scaled in himetric units rather than twips; see §6.14.7 [Preserving graphics scale in Word](#) on page 191.

Differences in how Word interprets RTF code are particularly noticeable in the following document features:

Graphics	Scaling units vary among Word versions; see §6.14.7 Preserving graphics scale in Word on page 191.
Tables	Straddled rows and rotated text are supported only by some Word versions; see §6.13 Converting tables to print RTF on page 184.
Links	Cross references and hypertext links can be made active and updatable in later versions of Word; see §6.11 Converting cross references and hypertext links on page 174.
Revision tracking	Revision tracking can be turned on automatically in some versions of Word; see §6.16 Turning on revision tracking in Word on page 194.

Multiple versions of Word

If your RTF output files will be viewed in multiple versions of Word, see §6.17.1 [Supporting more than one version of Word](#) on page 195.

6.4 Converting a FrameMaker book to print RTF

Word becomes unstable at much smaller document sizes than are typical for FrameMaker books. Combining FrameMaker files into a single RTF file usually results in a file that crashes Word. Word is limited to a maximum of about 150 pages per file, though you might get away with larger files, or fail with smaller files. So merging the (usually larger) FrameMaker files is a risky proposition. This is true especially if FrameMaker files contain graphics; Word keeps a local copy of every referenced graphic in the document file, so file sizes can become huge very quickly.

If you really must produce a single RTF file from a FrameMaker book, you can do so:

- in FrameMaker before conversion:

§2.5.6 [Producing a single output file from a FrameMaker book](#) on page 73

- in Word after conversion:

§6.17.4 [Combining RTF files into a Word master document](#) on page 197.

6.5 Specifying document layout options

You might have to experiment with page-layout settings to achieve correct spacing in Word. For most layout changes in FrameMaker, **Mif2Go** starts a new section in Word.

In this section:

§6.5.1 [Understanding page layout restrictions](#) on page 151

§6.5.2 [Eliminating large top or bottom margins](#) on page 151

§6.5.3 [Using text frames to solve spacing problems](#) on page 152

§6.5.4 [Maintaining pagination in Word](#) on page 152

§6.5.5 [Managing page and section breaks](#) on page 152

§6.5.6 [Specifying columns and gaps](#) on page 153

§6.5.7 [Adjusting sidehead width for Word](#) on page 153

§6.5.8 [Converting footnotes](#) on page 153

§6.5.9 [Converting headers and footers](#) on page 154

§6.5.10 [Converting special text flows for RTF output](#) on page 156

§6.5.11 [Handling different page size or orientation](#) on page 157

6.5.1 Understanding page layout restrictions

Word does not cope well with varying page layouts. While it is possible to alter the header and footer for the first page, it is not possible to alter other page layout features the way you can in FrameMaker. You must observe the following layout restrictions in your FrameMaker document:

- Use only **First**, **Left**, and **Right** master pages, and no others.
- Keep all master-page body frames exactly the same size, at the same vertical position. (Mirroring for left-right is acceptable; so is using a fixed gutter.)
- Make sure that no part of the header extends below the top of the body frame, and that no part of the footer extends above the bottom of the body frame.
- Keep the header and footer simple: only one text frame each, with any graphics in anchored frames.

If your FrameMaker document has a more complex layout, you might have to use a conversion template to create an intermediate layout that conforms to these restrictions; see §2.4 [Importing formats from a conversion template](#) on page 67.

6.5.2 Eliminating large top or bottom margins

An unusually tall top or bottom margin in Word is almost always caused by a master-page element in your FrameMaker document; see §6.1.1 [Understanding differences in implementation](#) on page 142. In Word, the body area must be entirely below the header and above the footer; this means that some FrameMaker layouts cannot be reproduced in Word.

The remedy is to use a FrameMaker conversion template; see §2.4 [Importing formats from a conversion template](#) on page 67. The template must include replacement master pages that do not have whatever element is causing the large top or bottom margin. Make sure

master pages are imported from the template, with a configuration setting; see §30.7.1 [Specifying conversion-template settings](#) on page 864.

Note: A Word template will not help with this problem.

6.5.3 Using text frames to solve spacing problems

When text flows through an extra frame, as in a lift or insert, **Mif2Go** tries by default to make a Word text box the same size, containing the text. When this works, it looks good. When it does not work, it makes a mess, and you are better off just having the content in the regular body text stream:

```
[WordOptions]
; UseTextFrames = Yes (default, to emulate framing) or No
UseTextFrames=Yes
```

Experiment with the following option when you are trying to solve bizarre spacing problems around text frames, such as when they push other text and frames far, far away:

```
[WordOptions]
; WrapAroundTextFrames = Yes (default, leave room around) or No
WrapAroundTextFrames=Yes
```

The default is to omit the RTF \nowrap property; when WrapAroundTextFrames=Yes, **Mif2Go** uses \nowrap.

6.5.4 Maintaining pagination in Word

Word copyfits more loosely than FrameMaker, and tends to break pages earlier. To maintain in Word the same pagination as in FrameMaker, you might have to create a conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67), and tune the template so page breaks occur at the same places in Word as in FrameMaker.

A conversion template might have to include format tweaking to maintain the same pagination in Word. Probably you would want to reduce default table cell margins, and space above and below tables, to make the tables in your document fit the same amount of space. Sometimes a reduction of 5% in the type size, such as changing 11 pt. to 10.5 pt. for Word output, brings the appearance of the document in Word closer to its appearance in FrameMaker. Also, a fractional reduction of paragraph spacing seems to help.

See also:

§6.5.5 [Managing page and section breaks](#) on page 152

6.5.5 Managing page and section breaks

You can have **Mif2Go** remove all page breaks, keep those specified via the paragraph format's Pagination setting (Top of: Page, Left Page, or Right Page), or keep all hard page breaks (the default). Normal keeps them all, including those inserted with overrides; Format does away with the overrides:

```
[WordOptions]
; PageBreaks =
; Normal (retain),
; Format (only in format def), or
; Remove
PageBreaks=Normal
```

Word uses section breaks to permit the kind of format changes FrameMaker handles with a new master page, such as a change of header/footer. When **Mif2Go** encounters a page (after the first) that does not use the default left and right master pages, **Mif2Go** starts a

new output section with that page, and sets a new first-page header and footer according to the FrameMaker master page used; see §6.5.9.4 [Converting headers and footers on different master pages](#) on page 155.

To prevent section breaks when the master page changes in FrameMaker:

```
[WordOptions]
; KeepSectBreaks = Yes (retain, Word default) or No (remove)
KeepSectBreaks=No
```

If you like what you get when KeepSectBreaks=Yes, use this setting; if not, set KeepSectBreaks=No.

6.5.6 Specifying columns and gaps

You can specify the number of columns to appear on an output page and the gap between columns. If you use the default values (zero), the number of columns and the size of the intercolumn gap stays the same as specified for the Right master page in FrameMaker:

```
[WordOptions]
; PageColumns = 0 (default, set per Right master) or number to use
PageColumns=0
; PageColGap = 0 (default, set per Right master) or twips to use
PageColGap=0
```

Mif2Go tries to determine reasonable values based on analysis of the master page text-frame geometry, which does not always come out right when you have a complex multi-flow layout.

6.5.7 Adjusting sidehead width for Word

If your FrameMaker document uses more than one master page, and one or more master pages specify different sidehead widths, tables that are oriented across all columns might appear distorted in Word output; this is because **Mif2Go** uses the sidehead width specified on the first master page for the entire document. If that width is inappropriate for the remainder of the document, you can specify a standard sidehead width in twips (twentieths of a point) for **Mif2Go** to use for the entire output.

For example, to specify a sidehead width of 1.1 inches and a gap of 0.15 inch:

```
[WordOptions]
; SHWidth = 0 (default, set per first usage) or twips to use
SHWidth=1584
; SHGap = 0 (default, set per first usage) or twips to use
SHGap=216
```

A better approach is to use a conversion template to establish a standard layout for Word output.

See also:

§2.4 [Importing formats from a conversion template](#) on page 67

§6.1.1 [Understanding differences in implementation](#) on page 142, [Sidehead formats](#)

§6.5.1 [Understanding page layout restrictions](#) on page 151

6.5.8 Converting footnotes

Word handles footnotes differently from FrameMaker. **Mif2Go** outputs FrameMaker asterisk footnotes as a count of asterisks. Subscripts and superscripts follow the FrameMaker document settings for relative size and position. The default configuration

setting for numbered footnotes uses fixed numbers, emulating FrameMaker's footnote handling: cross references to footnotes are correct, and the numbers stay the same.

If you need to add or delete footnotes in the output Word document, and you want auto-renumbering to work, you can take your chances with "real" Word footnotes:

```
[WordOptions]
; Footnotes = Standard (Word default), Embed (between []), None, or
; Variable (using real Word footnotes, so xrefs to them are wrong)
Footnotes=Variable
```

6.5.9 Converting headers and footers

Usually the best approach to converting headers and footers is to determine what you want to see in Word, then create FrameMaker content that produces that result on conversion, regardless of what the master page looks like in FrameMaker. Use the resulting master pages in a conversion template, which is a necessity for Word conversions involving complex header/footer usage.

In this section:

[§6.5.9.1 Understanding how Mif2Go treats header and footer text](#) on page 154

[§6.5.9.2 Converting headers and footers for WordPerfect](#) on page 154

[§6.5.9.3 Adjusting headers and footers for conversion](#) on page 154

[§6.5.9.4 Converting headers and footers on different master pages](#) on page 155

[§6.5.9.5 Positioning header and footer text and graphics](#) on page 155

[§6.5.9.6 Converting alternate running headers and footers](#) on page 156

[§6.5.9.7 Eliminating missing-formats error messages](#) on page 156

See also:

[§2.4 Importing formats from a conversion template](#) on page 67

[§30.7 Applying FrameMaker conversion templates](#) on page 863

6.5.9.1 Understanding how Mif2Go treats header and footer text

Mif2Go outputs the actual FrameMaker header and footer content (and any master-page graphics) in Word absolute-positioned frames, anchored in the Word header and footer areas. The only way to change header and footer content in Word is with a section break. For conversion to Word, your FrameMaker master pages can alter header and footer *content* for left, right, and first pages, but the header and footer *text frames* must be identical across all pages.

6.5.9.2 Converting headers and footers for WordPerfect

WordPerfect cannot handle RTF frames; for WordPerfect, **Mif2Go** writes FrameMaker header and footer text without frames. When you specify WordPerfect as the output type, **Mif2Go** automatically sets HeadFoot to Text and HFFramed to No:

```
[WordOptions]
; HeadFoot = Standard, Text (no graphics), or None (for WinHelp)
HeadFoot=Text
; HFFramed = Yes (default) to position headers/footers using frames
HFFramed=No
```

6.5.9.3 Adjusting headers and footers for conversion

Headers and footers often need adjustment for conversion to Word. For example:

- A running header or footer in Word can reference only one paragraph format, not a list of alternate formats.
- Master-page elements in Word cannot go down the side of the page; all header material must be above any body material, and all footer material must be below the body.

You might have to prepare a conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67) to apply before converting your document, to make headers and footers comply with the following requirements:

- Header elements must not extend below the top edge of the body frame.
- Footer elements must not extend above the bottom edge of the body frame.
- Top and bottom of the body frame must be in the same position on *all* pages (including the first).
- All header text must be in one header frame, and all footer text in one footer frame.
- Graphics must be in anchored frames, and must not extend outside the header or footer frame.

If the header extends downward or the footer extends upward (often along the side edge in FrameMaker), you might find that Word forces a page break, so that there is only one line per page. In Word, the body text has to start below the lowest point in the header, and end above the highest point in the footer. When that is not possible, Word puts out a minimum of one line anyway.

6.5.9.4 Converting headers and footers on different master pages

When the first page uses a master page different from the normal left and right, **Mif2Go** identifies the master page, creates the required header and footer (if used), and picks up any first-page graphics and untagged text frames.

Word uses section breaks to permit the kind of format changes FrameMaker handles with a new master page, such as a change of header/footer. When **Mif2Go** encounters a page (after the first) that does *not* use the default left and right master pages, **Mif2Go** starts a new output section with that page, and sets a new first-page header and footer according to the FrameMaker master page used. See §6.5.5 [Managing page and section breaks](#) on page 152.

6.5.9.5 Positioning header and footer text and graphics

By default, **Mif2Go** uses text frames to position headers and footers in the output. You can specify whether to let framed header/footer text wrap, and you can adjust the space around the frames:

```
[WordOptions]
; HFFramed = Yes (default) to position headers/footers using frames
HFFramed=Yes
; WrapAroundHFFrames = Yes (default, let text wrap around) or No
WrapAroundHFFrames=Yes
; HFGap = twips to space around header/footer frames, default none
HFGap=0
```

*Aligning headers
and footers with
graphics*

If you have graphic elements placed directly on the master page in the header and footer areas, the text in the header/footer text frames might not align vertically with those elements, because FrameMaker and Word calculate frame positions differently. To correct the positioning, you can set a vertical adjustment value (positive to move down, negative to move up) for the text. The default setting is 100 twips (twentieths of a point), which moves the text down 5 points:

```
[WordOptions]
; HFVertAdjust = twips to move header/footer text down
; (negative for up)
HFVertAdjust=100
```

*Eliminating
graphics from
headers and
footers*

If you do not want to retain FrameMaker header/footer graphics in Word, you can specify that headers and footers should be text only:

```
[WordOptions]
; HeadFoot = Standard (default), Text (no graphics),
; or None (for WinHelp)
HeadFoot=Standard
```

6.5.9.6 Converting alternate running headers and footers

Your FrameMaker document might use Running H/F variables that have alternate values, with definitions such as the following:

```
<$paratext[SectTitle,ChapTitle,AppxTitle]>
```

Word does not support alternate text for headers and footers, so **Mif2Go** uses the only the first format encountered in the definition. However, you can achieve the same effect in Word by directing **Mif2Go** to map all the formats to the same Word style. For example:

```
[Styles]
SectTitle=Title

[StyleReplacements]
ChapTitle=Title
AppxTitle=Title
```

The result is {STYLEREF Title} in Word. The mapping changes the style name, but not the format properties. If the format properties are different, *ChapTitle* and *AppxTitle* items would keep their FrameMaker properties as overrides, unless you were to apply a Word template to remove them. You can replace as many formats as you please this way.

6.5.9.7 Eliminating missing-formats error messages

If your headers or footers contain variables that refer to unused paragraph formats, the following setting prevents Word from displaying an error message:

```
[WordOptions]
; WriteMissingForms = Yes (default), write empty hidden paras
; at start of file for var formats that are used in headers/footers
; but not in doc)
WriteMissingForms=Yes
```

6.5.10 Converting special text flows for RTF output

In FrameMaker you might be using a special-purpose text flow that you do not want in the converted file. For example, you might use a different flow for starting autonumbers, in white text so they do not show. When you convert a file with multiple flows, the added flows come out as added sections, following the main text. This would be fine if the flow were for a sidebar, for example, where you want to keep the text. But if it is one you would rather lose, and it has a different Flow Tag from the text you do want to convert, you can direct **Mif2Go** to skip the unwanted text flow:

```
[TextFlows]
; flowtags to Skip
; or to treat as Normal (to keep in same RTF section)
z=Skip
A=Normal
B=Normal
```

You can also use this facility to put different flows in different RTF files, by running **Mif2Go** once to create each file, editing the configuration file to skip a different set of flows before each run.

If your FrameMaker document contains separate text flows for material you want included in the normal flow (such as boilerplate elements), you can make the special flow part of the normal flow by adding lines setting both flows to Normal.

Mif2Go normally processes each tagged text flow in the FrameMaker file as a separate section in the Word file. To make the content of all flows to appear in page sequence instead:

```
[WordOptions]
; SingleFlow = No (default, handle flows separately) or Yes
SingleFlow=Yes
```

6.5.11 Handling different page size or orientation

If the master pages in your document are of different sizes, and you are converting to Word, **Mif2Go** starts a new Word section each time the page size changes. For example, if your First master page has a body text frame that is a different size from the body text frames on Left and Right pages, in Word you will get a section break between first and second pages; probably not what you want. You have two choices:

- Use a conversion template that has all master-page body text frames the same size (see §2.4 [Importing formats from a conversion template](#) on page 67).
- Use paragraph spacing and Frame Above / Frame Below in FrameMaker to get the desired effect on the First page, but keep the body-text frame the same size as for Left and Right pages.

If you use landscape pages for large tables, as long as no text flows across the page breaks before and after each table, the effect should look the same in Word as in FrameMaker. A change to or from landscape can happen only at a section break in Word.

6.6 Converting system variables to text for RTF

The only way **Mif2Go** can get the text content of FrameMaker system variables (such as date and time) *that appear on body pages* into RTF output is to convert these variables to text. Other variables are already available in a usable form, in the MIF files.

To convert date/time and file-name system variables, that appear on body pages in FrameMaker, to text in the output, specify the following setting:

```
[Setup]
; ConvertVariables = No (default) or Yes (convert to plain text)
ConvertVariables=Yes
```

For example, you might want to use this setting to get the value of Creation Date or Modification Date to appear in the output.

Note: You do not need this setting for system variables *that appear on master pages*; **Mif2Go** converts those variables to their Word field equivalents. For Running H/F variables, see §6.5.9.6 [Converting alternate running headers and footers](#) on page 156.

Note: For system variables to show up in the MIF files, **Mif2Go** must read your original FrameMaker files. If you specify **Use existing MIF** on the *Export* dialog or in your project configuration file, system variables are not converted.

6.7 Converting paragraph and character formats

Sometimes differences in available formats between FrameMaker and Word can make precise conversions tricky. You might have to experiment with different settings to get the appearance you want.

In this section:

- §6.7.1 [Mapping paragraph formats to RTF styles](#) on page 158
- §6.7.2 [Merging paragraph formats](#) on page 159
- §6.7.3 [Converting sidehead formats](#) on page 159
- §6.7.4 [Converting run-in headings](#) on page 160
- §6.7.5 [Converting autonumbered formats](#) on page 160
- §6.7.6 [Converting bulleted formats](#) on page 162
- §6.7.7 [Converting reference frames for Word](#) on page 162
- §6.7.8 [Converting character formats](#) on page 163
- §6.7.9 [Removing unused formats](#) on page 163

6.7.1 Mapping paragraph formats to RTF styles

You can remap formats to have other names in the output file. One reason for doing this might be to enable the use of Word's paragraph autonumbering facility, which requires use of the predefined *Heading 1* through *Heading 9* styles:

```
[Styles]
; Document para format name = RTF style name (affects name only)
; Always use the remapped (RTF) name in the HelpStyles sections
; the RTF name must be unique; some examples are shown below
; the Heading N styles support Word outline and autonumber features
ChapTitle=Title
Heading1=Heading 1
Heading2=Heading 2
Heading3=Heading 3
Heading4=Heading 4
Heading5=Heading 5
HeadingRunin=Heading 6
Numbered1=Heading 7
Numbered=Heading 8
Heading9=Heading 9
```

*You cannot use
[Styles] to merge
formats*

Even though in FrameMaker the *Numbered1* and *Numbered* formats are really at the same level, they are *not* assigned the same name in the output file. Each style name used for the RTF file must be unique; you cannot use the [Styles] section to merge styles. If you try to do so, Word unmerges them for you, by adding a digit after the repeating name. For example, if you specify both *Numbered1=Heading 7* and *Numbered=Heading 7*, Word renames the second style *Heading 71*. Use the [StyleReplacements] section to map multiple formats to a single style; see §6.7.2 [Merging paragraph formats](#) on page 159.

*You cannot map
to Normal style*

For the same reason, avoid mapping any format to *Normal*, which Word uses for its first style. Word renames such mapped styles *Normal1*, *Normal2*, and so on.

*Null mappings are
ignored*

Mif2Go ignores any assignment in the [Styles] section that has no entry to the right of the equals sign.

6.7.2 Merging paragraph formats

To merge formats, use the [StyleReplacements] section to map each format to an existing RTF style. For example:

```
[StyleReplacements]
; Document para format name = replacement existing RTF style name
; the RTF name need not be unique, and may be created in [Styles]
; the properties of the original style remain as-is in the text
; unless Template and TemplateAutoUpdate are set in [WordOptions]
HeadingRunIn=Heading 2
SubHead=Heading 2
```

Mif2Go ignores any assignment in the [StyleReplacements] section that has no entry to the right of the equals sign.

6.7.3 Converting sidehead formats

If your document uses FrameMaker sideheads, you have some choices to make, because Word does not support sideheads as such. **Mif2Go** offers four options:

```
[WordOptions]
; Sideheads = Left (simplest), Indent, Frame (most accurate),
; or Normal
```

Options for Sideheads have the following effects:

Left	Sets the sideheads flush left in the main text column, which is widened to the full page width (less margins).
Indent	Uses left and right indents to create a sidehead-column effect. The sideheads are all right-indented, and the body is left-indented. Heads that span both columns continue to do so. The sideheads are aligned with their bottoms, instead of their tops, even with the top of their related text.
Frame	<i>(Not available for WinHelp)</i> Does an excellent emulation of FrameMaker sideheads most of the time, <i>provided the sideheads are always on the left</i> ; do not even try it if you have sideheads set to “Inside” or “Outside”. This option works by placing all sideheads in text-relative Word frames, and narrowing the text column to duplicate the FrameMaker layout. Mif2Go undertakes some serious reformatting to achieve this effect. For example, the Space Above for both the sidehead and its text paragraph must be set to 0; otherwise they will not align correctly if they happen to fall at the top of a page. Mif2Go changes the Space Below of the previous paragraph to compensate. The spanning heads are also in frames, anchored to a dummy paragraph in the text column so that they do not conflict with any following sideheads.
Normal	Uses whatever the paragraph format specifies, which is usually the wrong thing to do.

Fine-tune sideheads

You can specify additional fine-tuning options to position sideheads in the output document. If you specify a value other than No for ForceSideHeadPos, that value applies if you have also specified Indent or Frame for Sideheads:

```
[WordOptions]
; ForceSideHeadPos = No (default), Left, Right, Inner, or Outer
ForceSideHeadPos=No
```

Span all columns

If your document uses a FrameMaker paragraph format that spans all columns including sideheads, **Mif2Go** inserts an anchor paragraph, so the text box containing the spanning paragraph does not become entwined with the text box containing a directly following

sidehead. If there is always body text in between, **Mif2Go** anchors the text box for the spanner to the body paragraph. (Word does not handle correctly two text boxes anchored to the same place):

```
[WordOptions]
; SHSpannerAnchors = Yes (default,
; anchor paras after framed spanners)
SHSpannerAnchors=Yes
```

Align sideheads You can attempt to improve the way sidehead text boxes align to body paragraphs:

```
[WordOptions]
; SHVertAdjust = twips to move sidehead framed text down (neg for up)
SHVertAdjust=0
```

6.7.4 Converting run-in headings

Word does not support run-in headings as such; however, **Mif2Go** can emulate FrameMaker run-in headings in RTF output:

```
[WordOptions]
; RunInHeads = Runin (Word default) or Normal (help default)
RunInHeads=Runin
```

RunInHeads=Runin, the default value for Word output, produces what appear to be run-in headings in Word. However, if a run-in heading occurs at end-of-flow in a table cell or text frame, and therefore has no following paragraph, **Mif2Go** treats the run-in heading as a normal paragraph regardless of the value of RunInHeads.

6.7.5 Converting autonumbered formats

To successfully convert FrameMaker autonumbers, it is best to use FrameMaker paragraph numbering facilities consistently. For example, if you use <\$chapnum> for some paragraph formats in a given autonumber series, use it for all formats in that same series.

In this section:

§6.7.5.1 [Understanding autonumbering options in Word](#) on page 160

§6.7.5.2 [Converting autonumbers to Word SEQ fields](#) on page 161

§6.7.5.3 [Converting autonumbers via Word style sheet](#) on page 161

6.7.5.1 Understanding autonumbering options in Word

By default, **Mif2Go** converts FrameMaker autonumbers to text in Word output; this is consistent with the objective of providing read-only output for Word. See §6.1 [Converting to Word: a one-way street](#) on page 141.

Instead of converting autonumbers to plain text, you can have **Mif2Go** do either of the following:

[Convert autonumbers to Word SEQ fields](#)

[Replace autonumbers with Word native numbering.](#)

*Convert
autonumbers to
Word SEQ fields*

If you are migrating FrameMaker documents to Word instead of using Word only for review, converting FrameMaker autonumbers to Word SEQ fields gives you “live” autonumbers in Word. Unlike Word style sheet-based numbering, Word SEQ fields are stable, and can represent any FrameMaker autonumber sequence. You get a flexible numbering system in Word that gives you exactly what you want, rather than the limited options available from the Word *Bullets and Numbering* dialog. See §6.7.5.2 [Converting autonumbers to Word SEQ fields](#) on page 161.

Replace
autonumbers with
Word native
numbering

You can have **Mif2Go** apply a Word template that replaces FrameMaker autonumbers with Word style sheet-based autonumbers; however, this method is not supported, and not recommended. FrameMaker has much more reliable and flexible autonumbering than Word, and only in the most trivial cases can you duplicate FrameMaker numbering using Word native autonumbering. See §6.7.5.3 [Converting autonumbers via Word style sheet](#) on page 161.

6.7.5.2 Converting autonumbers to Word SEQ fields

To convert FrameMaker autonumbers to Word SEQ fields, set the following options:

```
[WordOptions]
; WriteAnums = Yes (default, write per SeqAnums) or No (omit entirely,
; used when having a Word style sheet add them for selected styles)
WriteAnums = Yes
; SeqAnums = No (default, write as text) or Yes (write as SEQ fields)
SeqAnums = Yes
```

When WriteAnums=Yes (the default) and SeqAnums=Yes, **Mif2Go** converts FrameMaker autonumbers to Word SEQ fields.

When WriteAnums=No, the value of SeqAnums is ignored, and FrameMaker autonumbers are omitted in favor of whatever Word stylesheet you supply. This is not a recommended option, because Word style-based numbering is very problematic.

If you find that non-numbered paragraphs also get numbered when you set WriteAnums=Yes and SeqAnums=Yes, look at the FrameMaker numbering properties of the format(s) in question. Sometimes people add numbering properties such as A:< =0> to the *Body* paragraph format, for example, to reset numbering for list formats; **Mif2Go** does not add numbering beyond what is specified in your FrameMaker document.

To eliminate unwanted autonumbers, you can import a FrameMaker conversion template to change the definitions of formats that should not be numbered in RTF output; see §2.4 [Importing formats from a conversion template](#) on page 67.

To insert a new item in a SEQ field-based autonumber sequence in Word, the best practice is to copy and paste an instance of that number style, either from the same sequence or from a Word template. You cannot include in a Word style SEQ fields (or bullets) *as such*.

6.7.5.3 Converting autonumbers via Word style sheet

Before attempting to use Word native autonumbering, consider the following:

- Many FrameMaker autonumber sequences cannot be represented this way.
- All FrameMaker-generated autonumbers in your document will be omitted from Word output, including autonumbers in cross-reference formats.
- Word-generated autonumbers might not be the same as the FrameMaker-generated autonumbers; the Word numbering system is notoriously buggy.
- Setting up a .dot template for style sheet-based autonumbers in Word is not a trivial exercise.
- **Mif2Go** does not support Word native autonumbering; if you use this method, you are on your own getting the numbers right using Word styles.

If you still want to take a chance with Word native autonumbering, make a Word template (a .dot file) with numbering set for the styles you wish; see §6.2.6 [Importing a Word template](#) on page 148. When you convert your document, **Mif2Go** attaches the .dot file to the Word .rtf file and makes it active.

In the **Mif2Go** configuration file, specify the following options:

```
[WordOptions]
; WriteAnums = Yes (default, write per SeqAnums) or No (omit entirely,
; used when having a Word style sheet add them for selected styles)
WriteAnums=No
; SeqAnums = No (default, write as text) or Yes (write as SEQ fields)
SeqAnums=No
```

When WriteAnums=No, **Mif2Go** omits FrameMaker autonumbers from RTF output, so Word style sheet autonumbers can be applied from a Word template. *Autonumbers also disappear from any cross references in your document.*

When WriteAnums=Yes, if you use Word autonumbering in your Word template, you will see both the Word autonumbers and the converted-to-text FrameMaker autonumbers in the output.

When you open the RTF file in Word, Select All (**Ctrl-A**) then update (**F9**), and the regular Word autonumbers should appear throughout the document. Thereafter, when you add a new numbered paragraph, the number appears automatically. You must use **Ctrl-A/F9** to get the numbers that follow to update; that is how style sheet-based numbering works in Word.

6.7.6 Converting bulleted formats

Bulleted lists in FrameMaker should convert to their RTF equivalents without a problem, except in the following circumstance:

- You are converting to Word 97.
- The FrameMaker format name for a bulleted paragraph starts with the word “bullet” followed by a space.

When a Word style has a name that starts with “bullet ”, Word 97 helpfully supplies a bullet as a prefix; the result is *two* bullets in front of each item. The problem does not occur in later versions of Word, and does not affect format names such as *Bulleted*.

If the RTF files you produce will be viewed in Word 97, you can rename the problem formats in the configuration file. For example:

```
[Styles]
Bullet Open=Bull Open
Bullet Sq=Bull Sq
Bullet Sq Indent=Bull Sq Indent
```

6.7.7 Converting reference frames for Word

You can choose whether or not to include reference frames defined for paragraph formats, and if so, whether to use the actual reference-page graphic or just its name. (You might want the name if it is descriptive, such as “Note” or “Caution”). If you specify Text to include just the name, also specify a format (FrameMaker paragraph format) for the name:

```
[WordOptions]
; RefFrames =
; Graphic (show Frame Above and Below),
; Text (name only), or
; None
RefFrames=Text
; RefFrameDefFormat = the format to be used for Text reference frames
RefFrameDefFormat=AlertHead
```

You can override the RefFrames settings for individual named reference frames:

```
[RefFrameFormats]
; override default RefFrames setting here for specific frame names
```



```
; give the frame name with no style, or "Graphic", after the "=" to
; keep the frame as a graphic, specify a style name to make it text,
; or use "None" to eliminate it altogether
Hints=Title 2
```

6.7.8 Converting character formats

Mif2Go can output FrameMaker character formats as Word character styles, instead of as overrides. This is sometimes difficult for Word to handle, however, so use this feature only if you anticipate needing to revise the Word text:

```
[WordOptions]
; CharStyles sets char properties in styles (causes problems with WP)
CharStyles=Yes
; CharStylesUsedInText = No (default); or Yes, use cs codes in text
CharStylesUsedInText=Yes
```

*Style names only,
not properties*

Be aware that if **Mif2Go** does not output style properties explicitly, they do not appear in Word. That is, the style name appears, but the actual properties are all Word defaults. That is why **Mif2Go** puts out what in FrameMaker would be overrides: there is no choice, in Word.

6.7.9 Removing unused formats

Mif2Go keeps track of which paragraph and character formats are actually used in the output file, and can remove unused formats. The default is to leave them in:

```
[WordOptions]
; RemoveUnusedStyles = No (default for Word) or Yes
RemoveUnusedStyles=No
```

If your document contains variables that refer to unused paragraph formats, the following setting prevents Word from displaying an error message:

```
[WordOptions]
; WriteAllVarForms = No (default) or Yes (write empty hidden paras
; for all var formats even if they appear to be used in doc body)
WriteAllVarForms=Yes
```

6.8 Converting tabs and spaces

Mif2Go uses font metrics (see [Table 6-3](#) on page 167) to convert from FrameMaker's absolute tabs to Word's relative tabs. **Mif2Go** uses these metrics to compute tabs in graphics as well as in text.

In this section:

- §6.8.1 [Understanding differences in tab behavior](#) on page 163
- §6.8.2 [Understanding differences in spaces](#) on page 164
- §6.8.3 [Altering tab behavior for Word output](#) on page 164
- §6.8.4 [Altering font metrics to adjust tabs](#) on page 165

6.8.1 Understanding differences in tab behavior

FrameMaker uses *absolute tabs*; Word uses *relative tabs*:

Absolute: Tab stops are specified at fixed positions; if the line before a tab runs past the tab position, the tab has no effect.

Relative: Wherever you are in a line, a tab causes a move to the next tab stop. Add or delete a few letters, and the text at the end (after all the tabs) can jump to the wrong columns.

To determine how many relative tabs to put out for RTF, **Mif2Go** must know two things:

- the intended tab position
- the current tab position.

Mif2Go stores the intended position. The current position must be computed, based on character size, character spacing, expansion and compression, and so forth.

Font metrics Computing character size for tab spacing is problematic. Because the combined font-metric information for all available fonts would exceed the capacity of many hard drives, **Mif2Go** simplifies font metrics by using PostScript relative-character-width rules and a single number, which you can specify in the configuration file; see §6.8.4 [Altering font metrics to adjust tabs](#) on page 165.

Unused tabs If a FrameMaker paragraph format (for example, *Header 1*) includes unused tabs, and the text before an unused tab ends very close to the position of that tab, **Mif2Go** might calculate that the text extends beyond the tab, while Word calculates that it stops before the tab, and therefore uses the “wrong” tab. The best way to correct this problem is to remove the unused tab from the FrameMaker format. The other remedy is to import a FrameMaker conversion template that defines the format in question without the unused tab; see §2.4 [Importing formats from a conversion template](#) on page 67.

6.8.2 Understanding differences in spaces

Unlike FrameMaker, Word does not have a “thin space”. The smallest space in Word is a nonbreaking space, so **Mif2Go** converts each thin space in your FrameMaker document to a nonbreaking space in Word. However, a Word nonbreaking space is twice the width of a FrameMaker thin space. If you include a series of thin spaces followed by a tab, for example in an autonumber format, the converted autonumber can extend past the tab setting; the result is that the tab is dropped in Word (see §6.8.1 [Understanding differences in tab behavior](#) on page 163).

Thin spaces can affect Word tabs To avoid overrunning tabs in Word, either reduce the number of thin spaces (for example, in an autonumber format change `\st\st\st` to `\st\st` or to `\sn\st`), or set the following option to force a space between the number and the following text:

```
[WordOptions]
OccludedTabs=Space
```

See also:

§6.7.5 [Converting autonumbered formats](#) on page 160

§6.8.3 [Altering tab behavior for Word output](#) on page 164

6.8.3 Altering tab behavior for Word output

You can modify some aspects of tab behavior in Word:

[Tab width](#)

[Hidden tabs](#)

[Right tabs](#)

[Trailing tabs](#)

[Underlined tabs](#)

[Comparing underlined vs. trailing tabs.](#)

<i>Tab width</i>	<p>You can specify a default tab width in twips (twentieths of a point):</p> <pre>[WordOptions] ; DefTabWidth = 0 (default, ignore undefined tabs) ; or twips (720 for 0.5") DefTabWidth = 0</pre>
<i>Hidden tabs</i>	<p>You can choose to remove tabs that have no effect in a FrameMaker paragraph (because their nominal ruler positions are hidden by text), replace each with a space, or keep them in the output document:</p> <pre>[WordOptions] ; OccludedTabs = Remove (normal), Space, or Tab OccludedTabs = Remove</pre>
<i>Right tabs</i>	<p>You can specify how to treat right-alignment tabs:</p> <pre>[WordOptions] ; RMarginTabs = ; Left (return), ; Right (keep in col), ; Both (default), ; None RMarginTabs = Both</pre>
<i>Trailing tabs</i>	<p>To preserve trailing tabs; for example, if you use a trailing tab to insert a signature line:</p> <pre>[WordOptions] ; TrailingTabs = No (default, omit trailing tabs) or Yes ; (preserve trailing tabs in Word output) TrailingTabs = Yes</pre>
<i>Underlined tabs</i>	<p>If you have underlined text in FrameMaker, and you tab within the text, FrameMaker does not continue the underline under the tabbed area (Word works the opposite way: underlines continue). By default, Mif2Go turns off the underline for the tabbed areas in Word output, which is not what you want if you are making up a form, with ruled lines made from the underlines.</p> <p>To underline tabbed areas that follow underlined text:</p> <pre>[WordOptions] ; UnderlineTabs = No (default, normal FrameMaker behavior) or Yes UnderlineTabs = Yes</pre>
<i>Comparing underlined vs. trailing tabs</i>	<p>These two settings (TrailingTabs and UnderlineTabs) are entirely different. Normally, at the end of a paragraph, Mif2Go simply cancels any leftover tabs. With TrailingTabs, Mif2Go writes them instead. With UnderlineTabs, Mif2Go does not write them.</p> <p>To emulate FrameMaker behavior, Mif2Go turns off the underline just before the (first) tab, unless UnderlineTabs is set. TrailingTabs has no effect on that action.</p>

6.8.4 Altering font metrics to adjust tabs

You might need to adjust font metrics if you find that the effect of tabbing in the RTF file does not match the effect in FrameMaker. The **Mif2Go** font metric value is an approximation; if the metric:

- underestimates text width, the result might be an extra tab
- overestimates text width, a needed tab might not be put out.

Missing tabs, extra tabs

Generally only very small adjustments (one unit at a time) are needed:

- If a tab is missing that should be present, reduce the metric value for the font used for that line.
- If an extra tab is present, increase the metric.

To specify font metrics:

```
[FontWidths]
; RTF font name = relative width, to compute line length for tabs
; reduce if needed tabs omitted, increase if extra tabs present
; add entries for any additional fonts used in RTF documents
Times New Roman=90
```

Table 6-3 on page 167 lists the default metrics for common fonts.

Note: Altering font metrics has no effect on the output of normal text; the metrics are used only to calculate tab positions, and to process text lines in graphics that include font changes.

*Use tables
instead*

The font metric value does not affect tabs at the start of a line, but only tabs positioned after some text. Often the best way to avoid trouble for such tabs is to use a FrameMaker table instead.

Another way to avoid trouble is to ensure the following:

- only tab stops that are actually used are present
- no tab stop is “too close” to the end of text in the previous column.

Worst case is when you have tabs set every half inch all the way across the page, with tight columns; you might not be able to find font-metric values that makes them all work.

6.9 Specifying font usage

In this section:

§6.9.1 [Setting default font parameters](#) on page 166

§6.9.2 [Remapping fonts](#) on page 166

§6.9.3 [Specifying font types](#) on page 167

§6.9.4 [Specifying font encoding for non-Western characters](#) on page 168

§6.9.5 [Specifying font encoding for FrameMaker 8 Unicode](#) on page 169

§6.9.6 [Removing unused fonts](#) on page 170

6.9.1 Setting default font parameters

You can specify a default output font, and set default height and width values for the font:

```
[Defaults]
FontName=Times New Roman
; FontSize is in twips, 240 = 12pt
FontSize=220
; FontWidth is width in twips for a 12pt en space
FontWidth=144
```

6.9.2 Remapping fonts

You can remap fonts the same way you remap formats. The default values map common FrameMaker PostScript fonts to their TrueType equivalents:

```
[Fonts]
; Document font name = RTF font name
; for winhelp, stick to TT fonts in the RTF
; MS Serif and MS Sans Serif are always available,
; but only at 8, 10, 12, 14, 18, and 24pt sizes
; these change from PS fonts to the corresponding TT:
Helvetica-Narrow=Arial Narrow
Helvetica=Arial
```

```

Times*=Times New Roman
Courier=Courier New
Century Schoolbook=NewCenturySchlbk
Common Bullets=CommonBullets
MyriaMM*=Arial

```

You can add to this list as you please; the names on the left must be unique, but those on the right can appear any number of times; that is, you can map several FrameMaker fonts to a single RTF font. You can use wildcards: ? to match any one character, and * to match zero or more characters. This can be useful when you have several font variations that start with the same characters, that should be remapped the same way; for example, MyriaMM*=Arial. You can use this feature even if some names contain * or ?, because Mif2Go always tries for an exact character match first.

6.9.3 Specifying font types

Font types are critical to correct Word handling of characters in fonts that FrameMaker considers to be “symbol”, but that are not among the common symbol fonts (Symbol, WingDings, and Zapf Dingbats). This font-type information is essential in the RTF font list; Word might produce errors if the information is incorrect. Mif2Go provides default font types for the fonts listed in [Table 6-3](#).

Table 6-3 Default font types and metrics for RTF

Font	Type	Metric
Arial	2	138
Arial Narrow	2	115
AvantGarde	2	144
Bookman	1	150
Bullets	5	180
CommonBullets	5	180
Courier	3	144
Courier New	3	144
Dingbats	5	180
Garamond	1	145
Helvetica	2	138
Helvetica-Black	2	155
Helvetica-Light	2	130
Helvetica-Narrow	2	115
Korinna	1	140
MS Serif	1	144
NewCenturySchlbk	1	135
Palatino	1	141
Symbol	6	130
Times	1	134
Times New Roman	1	120
Webdings	5	180
Wingdings	5	180
ZapfChancery	4	121
ZapfDingbats	5	180

Mif2Go uses the font metrics shown in [Table 6-3](#) to convert tabs; see §6.8.4 [Altering font metrics to adjust tabs](#) on page 165.

If your document contains fonts other than those listed in [Table 6-3](#), you can determine their types as follows:

1. In Word, create a document that uses those fonts.
2. Save the document as RTF.
3. Look at the RTF in a text editor to find the RTF type number and name associated with each font. The font table is at the start of the RTF, and looks something like this:

```
{\fonttbl
{\f1 \froman \fcharset0 Times New Roman;}
{\f2 \fswiss \fcharset0 Arial Rounded MT Bold;}
{\f3 \fswiss \fcharset0 Arial;}
{\f4 \fmodern \fcharset0 Courier New;}
{\f5 \fswiss \fcharset0 Arial Narrow;}
{\f6 \fttech \fcharset2 Symbol;}
{\f7 \fdecor \fcharset2 Wingdings;}}
```

The `\fN` number at the beginning of each entry is the type number. [Table 6-4](#) shows the type number, type name, font family, and character-set encoding for each font type.

Table 6-4 RTF font types and font families

Type	Type name	Font family	Encoding
0	fnil	Unknown or default fonts	fcharset0
1	froman	Roman, proportionally spaced serif fonts	fcharset0
2	fswiss	Swiss, proportionally spaced sans serif fonts	fcharset0
3	fmodern	Monospaced serif and sans serif fonts	fcharset0
4	fscript	Script fonts	fcharset0
5	fdecor	Decorative fonts	fcharset2
6	fttech	Technical, symbol, and mathematical fonts	fcharset2
7	fbidi	Arabic, Hebrew, and other bidirectional fonts	fcharset0

4. Specify the font name and type number in the configuration file. For example:

```
[FontTypes]
; Font name = type number (0 to 7)
Arial Rounded MT Bold=2
```

See also:

§6.9.4 [Specifying font encoding for non-Western characters](#) on page 168

§6.9.5 [Specifying font encoding for FrameMaker 8 Unicode](#) on page 169

6.9.4 Specifying font encoding for non-Western characters

To produce RTF output that includes non-Western characters (text in which the “high ASCII” character set contains characters other than the European accented characters), you might have to specify an encoding value for a Windows font that contains the characters you need at the same code points as the font you use in FrameMaker.

Note: If you are using FrameMaker version 8.x, see §6.9.5 [Specifying font encoding for FrameMaker 8 Unicode](#) on page 169, for methods to convert Unicode characters to the Windows encodings needed by Word.

The RTF font table (see §6.9.3 [Specifying font types](#) on page 167) shows an encoding value for each font, as `fcharsetN`. The value of *N* for the Western character set is 0 (zero) for font types 1 through 4 and font type 7 (see [Table 6-4](#) on page 168). To determine the correct value of *N* for your language, use a text editor to examine the RTF font-table entries for a file that Word renders correctly in your language. Some `fcharsetN` values:

<u>Character set</u>	<u>fcharsetN</u>
Baltic	186
Central European	238
Cyrillic	204
Greek	161
Johab	130
Mac	77
Thai	222
Turkish	162
Vietnamese	163

For example, to specify Cyrillic font encoding:

```
[FontEncoding]
; Font name = value to use in font table for fcharset
Times New Roman CYR = 204
```

This setting tells Word to display Cyrillic characters in place of accented characters for the high ASCII code points.

Sometimes Word gets it right without this clue, but it never hurts to be explicit. **Mif2Go** simply passes the number through to the RTF font table, so you can use any number that you find works for you in Word (or in WinHelp).

See also:

§6.9.3 [Specifying font types](#) on page 167

6.9.5 Specifying font encoding for FrameMaker 8 Unicode

For FrameMaker version 8.x Unicode, at present **Mif2Go** supports Cyrillic 204, Central European 238, Greek 161, and Turkish 162 character sets. The font encoding for these character sets for Times New Roman, Arial, and Courier New is built in for FrameMaker 8 conversions, and need not be specified. For Asian fonts, the supported encodings are 128 for Japanese (Shift-JIS), 136 Traditional Chinese, 134 Simple Chinese, and 129 Korean.

For other conversions from FrameMaker 8 Unicode, the default fonts used for characters that are not in the Windows ANSI encoding are as follows:

<u>Font type</u>	<u>Font used</u>
froman	Times New Roman
fswiss	Arial
fmodern	Courier New

In each case, the encoding identifier (Cyr, CE, Greek, or Tur) is added to the font name.

To use different fonts for one or more of font types `froman`, `fswiss`, or `fmodern`, map them to the default fonts. For example:

```
[DefaultUnicodeFonts]
Times New Roman Cyr = Century Cyr
```

Make sure the font you specify really contains the characters for the encoding you indicate.

See also:

§6.9.3 [Specifying font types](#) on page 167

§6.9.4 [Specifying font encoding for non-Western characters](#) on page 168

6.9.6 Removing unused fonts

Mif2Go keeps track of which fonts are actually used in a file, and can remove unused fonts:

```
[WordOptions]
; RemoveUnusedFonts = No (default for Word) or Yes
RemoveUnusedFonts = Yes
```

Generally, you would want to keep all fonts in a document, so the Word default is to leave them in.

6.10 Modifying text appearance

In this section:

§6.10.1 [Adjusting line spacing](#) on page 170

§6.10.2 [Adjusting paragraph spacing](#) on page 170

§6.10.3 [Adjusting small caps](#) on page 172

§6.10.4 [Specifying a style for quotes](#) on page 172

§6.10.5 [Mapping high ASCII characters for RTF output](#) on page 172

§6.10.6 [Specifying text color](#) on page 172

§6.10.7 [Hiding white text](#) on page 173

§6.10.8 [Hiding content in Word](#) on page 173

§6.10.9 [Omitting content from RTF output](#) on page 174

§6.10.10 [Replacing content in RTF output](#) on page 174

6.10.1 Adjusting line spacing

The default setting for line spacing allows Word to expand lines to accommodate taller characters, such as embedded bitmaps. If you prefer that Word use exact line spacing, set the following option:

```
[WordOptions]
; ExactLineSpace = No (default, variable line height allowed) or Yes
ExactLineSpace=Yes
```

You can also set a default line space for the output, in twips (twentieths of a point):

```
[Defaults]
; LineSpacing is in twips, 240 = 12pt
LineSpacing=240
```

6.10.2 Adjusting paragraph spacing

In this section:

§6.10.2.1 [Understanding inter-paragraph spacing differences](#) on page 171

§6.10.2.2 [Keeping or reducing space above paragraphs](#) on page 171

§6.10.2.3 [Keeping or removing empty paragraphs](#) on page 171

§6.10.2.4 [Adjusting paragraphs in extra text frames](#) on page 172

6.10.2.1 Understanding inter-paragraph spacing differences

Word inter-paragraph spacing for versions of Word up to Word 2003 was additive: the space between two paragraphs was equal to the space below the first paragraph *plus* the space above the second paragraph. FrameMaker inter-paragraph spacing equals the space below the first paragraph *or* the space above the second paragraph, whichever is greater. To make Word and FrameMaker inter-paragraph spacing equivalent, you would have to set space above to zero for all paragraph formats in FrameMaker, and just use space below.

In addition, FrameMaker cancels the space above the first paragraph in a frame, such as at the top of a page; Word does not. Again, using zero space above is the only way to get the equivalent in Word; but then you have a problem with headings that do *not* start new pages.

You might think you could import a Word template to correct spacing for selected paragraph styles (see §6.2.6 [Importing a Word template](#) on page 148). However, applying a Word template does not remove overrides.

6.10.2.2 Keeping or reducing space above paragraphs

By default, **Mif2Go** adds overrides in Word to make spacing come out the same as in FrameMaker; that is, **Mif2Go** reduces space before in the RTF output (though not in the style definition) to simulate FrameMaker's larger-of-two-values rule instead of Word's sum-of-two-values rule.

To direct **Mif2Go** to use the Word sum-of-two-values rule instead of the FrameMaker rule for inter-paragraph spacing:

```
[WordOptions]
; ParaSpace = Normal (retain above & below) or Frame (adjust above)
ParaSpace=Normal
```

When ParaSpace=Normal, **Mif2Go** uses both space above and space below in RTF output. This can result in overlarge spacing.

When ParaSpace=Frame (the default), **Mif2Go** checks adjacent paragraphs case by case, and alters the space above the second paragraph of each pair (often to zero) as an override, in an attempt to match Word spacing to FrameMaker spacing. However, an exact match is not always possible.

6.10.2.3 Keeping or removing empty paragraphs

By default, **Mif2Go** removes empty paragraphs. If you use empty paragraphs in FrameMaker for their spacing effect, and the result in Word is insufficient space without them, you can specify keeping the anchors:

```
[WordOptions]
; KeepAnchorParagraphs = No (default),
; Yes keeps anchor paras as spacers
KeepAnchorParagraphs=Yes
```

To keep empty paragraphs above selected paragraph formats only, instead of setting KeepAnchorParagraphs=Yes you could use a macro to put the empty paragraph back in the RTF output just for those formats. For example:

```
[WordStyles]
ChapTop = CodeBefore

[ParaStyleCodeBefore]
ChapTop = \pard \s0 \sa240 \par
```

These settings would put an empty paragraph before each *ChapTop* paragraph in RTF. The RTF code `\sa240` would add 240 twips (twentieths of a point) of space below the empty paragraph. You can adjust this number to suit. See §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

6.10.2.4 Adjusting paragraphs in extra text frames

If you have a text frame on a body page other than the normal body frame (often the case on a “First” page when the title is positioned using such a frame), and there is more than one paragraph in the frame, the paragraphs might wind up on top of each other. If the massive chunk of **Mif2Go** code that tries to detect and fix this fails, you can give this setting a shot:

```
[WordOptions]
; UseParaAnchors = No (default),
;   Yes for successive paras in same frame
UseParaAnchors=Yes
```

6.10.3 Adjusting small caps

Word sometimes has difficulty producing text in Small Caps style. The text might be too small to read easily, reducing the size of the original caps. If this happens, you can turn off the Small Caps style. Or, you can make the Small Caps in Word follow the FrameMaker size rules:

```
[WordOptions]
; SmallCaps = Standard (default), Frame (using FM sizing), or None
SmallCaps=Frame
```

6.10.4 Specifying a style for quotes

Converting to Word for print output, you might choose to use keep FrameMaker “smart quotes” (curly quotes); or, set Quotes to Help to convert them all to vertical quote marks:

```
[WordOptions]
; Quotes = Help (only " and ') or Standard (allow left/right quotes)
Quotes=Standard
```

6.10.5 Mapping high ASCII characters for RTF output

Changing the following option to Yes suppresses the remapping of high-bit-set ASCII characters (typically accented characters) from FrameRoman, forcing their literal use. If you find that such characters are turning into unwanted different characters in the output, setting NoSymMap=Yes can fix the problem; if not, setting NoSymMap=Yes will cause the problem instead:

```
[WordOptions]
; NoSymMap = No (default, map text chars from Frame set)
;   or Yes (unmapped)
NoSymMap=No
```

6.10.6 Specifying text color

Mif2Go converts FrameMaker color definitions from CMYK to RGB, and adds them to the RTF color table so that they look the same in Word. You can choose whether to retain text colors:

```
[WordOptions]
; TextColor = 0 (all black) or 1 (as is, the default)
TextColor=1
```

If you use text colors only for ease of editing in FrameMaker, and want them to come out black in the converted files, set `TextColor=0` (this is the default for WinHelp conversions.) You can keep text colors as they are, with `TextColor=1`.

Note: The `TextColor` setting does not affect the use of color in graphics, including graphics text (such as callouts). The RTF color table also does not affect the colors used in graphics; color information is embedded in the graphic metafile.

6.10.7 Hiding white text

One way to put “invisible” text into FrameMaker is to make the text white, so that FrameMaker does not display the text unless it is against a dark background. To make **Mif2Go** ignore white text, so that paragraphs containing *only* white text are removed entirely, set this option to `Yes` (the default is `No`):

```
[WordOptions]
; HideWhiteText removes any white text (standard FrameMaker behavior)
HideWhiteText=Yes
```

If your FrameMaker document contains any white text you want to retain, use the default setting `HideWhiteText=No`, and get rid of unwanted instances of white text by one of the following methods:

- Assign their formats the `[WordStyles]Delete` property (see §6.10.9 [Omitting content from RTF output](#) on page 174).
- Apply FrameMaker conditional text to hide them.

6.10.8 Hiding content in Word

To convert content so it becomes hidden text in Word, a unique paragraph format for the material in FrameMaker, and assign the following property to the format:

```
[WordStyles]
; Hide makes the content Word hidden text.
ParaFmt=Hide
```

When you assign the `Hide` property to a paragraph format, all text in that format remains hidden in Word unless the user chooses to view hidden text. This setting is meant for Word output only; the `Hide` property is ignored in WinHelp.

If you really want the content not to be included at all in Word, see §6.10.9 [Omitting content from RTF output](#) on page 174.

*Anchor at
insertion point*

If content to be hidden includes graphics or other items in anchored frames, the frame anchors must be placed “at insertion point” in the anchoring paragraph. If you give an anchored frame any other position, its contents remain visible in Word.

*Mif2Go uses
hidden text*

Mif2Go uses Word hidden text to emulate several FrameMaker features. If you have other plans for hidden text in Word, and you do not want **Mif2Go** codes visible in Word when a user views hidden text, you must disable the following conversion features:

- Word-generated index; see §6.17.2 [Including index terms in Word](#) on page 195.
- Live cross references; see §6.11.2.2 [Making cross references active and updatable](#) on page 175.
- Live page numbers in interfile links; see §6.11.5.3 [Making page numbers in interfile links updatable](#) on page 180.

6.10.9 Omitting content from RTF output

To prevent text from appearing in RTF output, you can use conditional text, or you can do the following:

1. Use a special value of @outputclass (for example, onlineonly)
1. Use a special paragraph format (for example, *OnlineOnly*) for all instances of the unwanted text in your document.
2. In the configuration file, assign property Delete to the paragraph format:

```
[WordStyles]
; format (char or para) = keyword
; Delete is used to remove displayable text
OnlineOnly=Delete
```

If a table or graphic is anchored in a paragraph whose format is assigned the Delete property, the table or graphic is retained, and only the text of the paragraph is deleted.

To remove unwanted blank paragraphs at the end of topics:

```
[WordOptions]
; FrameEndPara = Yes (default, preserve empty paragraph at end of text
; frame) or No (remove empty final paragraph)
FrameEndPara = No
```

6.10.10 Replacing content in RTF output

This method is deprecated in favor of the CodeReplace property described in §28.9.3 Surrounding or replacing text with code or macros on page 822.

You can direct **Mif2Go** to replace the content of a paragraph, or of a character-formatted span of text, with arbitrary RTF code:

```
[WordStyles]
; format (char or para) = keyword
; Replace deletes, and also puts out the RTF in [WordReplacements]
Parafmt = Replace
```

You specify the replacement RTF code as a property of the format to which you assigned the Replace property:

```
[WordReplacements]
; Replace causes the insertion of raw RTF code
; in place of the original content of the named para or char format
Parafmt = {some arcane string of RTF code}
```

See also:

§6.10.2.3 [Keeping or removing empty paragraphs](#) on page 171

§28.3.7.2 [Inserting code with the CodeStore property](#) on page 804.

6.11 Converting cross references and hypertext links

In this section:

§6.11.1 [Including ObjectIDs for Word links and cross references](#) on page 175

§6.11.2 [Converting cross references to Word](#) on page 175

§6.11.3 [Converting hypertext links to Word](#) on page 178

§6.11.4 [Locking hypertext links to allow revision tracking](#) on page 178

§6.11.5 [Enabling interfile cross references and hypertext links](#) on page 179

§6.11.6 [Replacing building blocks in master-page references](#) on page 181

6.11.1 Including ObjectIDs for Word links and cross references

For active cross references and hypertext links in Word output, FrameMaker ObjectIDs must be included in RTF output.

To specify whether **Mif2Go** should include the FrameMaker ObjectIDs from your document:

```
[WordOptions]
; ObjectIDs = Referenced (default, used by TOC or IX), None, or All
ObjectIDs=Referenced
```

ObjectIDs values have the following effects:

Referenced	(Default) Mif2Go includes in the output every ObjectID in your document that serves as a target of a link.
None	FrameMaker ObjectIDs are omitted from the output. The result is that no links are active in Word.
All	Every ObjectID in your document ends up in the output.

The default setting allows you to have active contents, index, cross-reference, and hypertext links in Word.

For information about ObjectIDs in FrameMaker, see §5.3 [Identifying files and objects](#) on page 117.

6.11.2 Converting cross references to Word

Cross references can become active links in Word, and they can be updated in Word.

In this section:

- §6.11.2.1 [Understanding how Mif2Go converts cross references](#) on page 175
- §6.11.2.2 [Making cross references active and updatable](#) on page 175
- §6.11.2.3 [Weighing cross-reference behavior trade-offs](#) on page 177
- §6.11.2.4 [Producing numeric vs. full-text cross references](#) on page 177
- §6.11.2.5 [Omitting cross references from RTF output](#) on page 177

6.11.2.1 Understanding how Mif2Go converts cross references

When you specify Word output, cross references in your FrameMaker document become Word bookmarks and bookmark-references, precisely emulating the way FrameMaker uses <\$paratext>, <\$paranum>, and <\$paranumonly>; Word has no direct equivalents for these constructs. If you subsequently make changes to the text or number of a source paragraph in Word, the cross reference also changes, after you update all fields.

By default, **Mif2Go** also wraps cross references as Word hypertext links.

6.11.2.2 Making cross references active and updatable

For Word output, by default **Mif2Go** converts cross references to hypertext links, then makes them act like cross references implemented as Word bookmarks:

```
[WordOptions]
; Xrefs = Standard (default, working), or None (plain text)
Xrefs=Standard
; XrefHyper = Yes (default, make xrefs work as hyperlinks) or No
XrefHyper=Yes
; LockXrefs = Yes (default, faster load)
```

```
; or No (allow updating of xrefs)
LockXrefs=Yes
```

The default values for these settings have the following effects:

- Xrefs=Standard** Changing the source text in Word, then updating the cross-reference field, changes the text in the reference.
- XrefHyper=Yes** Clicking the reference in Word (**Ctrl**-clicking in Word 2003) takes you to the source of the reference.
- LockXrefs=Yes** Updating a cross reference to reflect changes to the source text requires unlocking the reference first.

When **Xrefs=None**, cross references are converted to text, and are not updatable; but if **XrefHyper=Yes**, they work as clickable links.

When **XrefHyper=No**, cross references do not work as links; but if **Xrefs=Standard**, they can be updated to match the source text.

When **LockXrefs=No**, you can update all cross references in Word without unlocking them first. However, what you lose is accurate page numbers in references. If your cross references do not include page numbers, this does not matter. If your cross references do include page numbers, there is little to gain by setting **LockXrefs=No**, because all you save in Word is a single click to unlock a reference for updating.

[Table 6-5](#) summarizes the effects of these settings.

Table 6-5 Effects of cross-reference settings in Word

Configuration settings			Cross references in Word		
Xrefs=	XrefHyper=	LockXrefs=	Updatable?	Active link?	Page #s OK?
Standard	Yes	Yes	Unlock first	Yes	Yes
		No	Yes	Yes	No
	No	Yes	Unlock first	No	Yes
		No	Yes	No	No
None	Yes	Yes	No	Yes	Yes
		No	No	Yes	No
	No	Yes	No	No	Yes
		No	No	No	No

Note: The former **FieldHyper** setting is deprecated (and is replaced by **XrefHyper**), but still works; the former **Xrefs=Fields** is replaced by **Xrefs=Standard**.

*References
between files*

If you are converting more than one FrameMaker file (typically a FrameMaker book), and there are cross references between files, to make those cross references work as expected you might need additional settings; see §6.11.5 [Enabling interfile cross references and hypertext links](#) on page 179.

*Referencing page
numbers in
different files*

Word does not support page references for cross-reference destinations in a different file, so **Mif2Go** produces a workaround in Word output. If you display hidden text in Word after converting cross references, immediately following each heading you will see a (hidden) page number; this is a Word **PAGEREF** field that points to the main bookmark for the heading. The **PAGEREF** field is bookmarked also. This way, references to the heading from other files can include dynamically updated page numbers.

6.11.2.3 Weighing cross-reference behavior trade-offs

Every version of Microsoft Word seems to behave differently with respect to cross references. To ensure accurately displayed cross references, you might have to target a single version of Word, and you might have to give up one or more of the following:

- revision tracking
- updating cross references
- using cross references as live links.

Wrapping cross references as hypertext links avoids some problems, causes others:

```
[WordOptions]
; WrapXrefs = Yes (default, wraps full xref format content as
; hyperlink, but displays xrefs as errors on Word 2000) or No
WrapXrefs=Yes
```

When WrapXrefs=Yes, cross references look and work as expected, but updating an interfile reference might cause a cannot-open-file error message to replace the text of the reference in Word 2000, even though the referenced file is present.

When WrapXrefs=No and LockXrefs=Yes, cross references are not active.

When WrapXrefs=No and LockXrefs=No, only the following cross references are active:

- Word 2000: cross references to target markers within the same file.
- Word 2003 and later versions: cross references to target markers in other files.

6.11.2.4 Producing numeric vs. full-text cross references

By default, cross references use FrameMaker numeric ObjectIDs instead of the full cross-reference text; full-text cross references are not active in Word:

```
[WordOptions]
; XrefType = = Numeric (default) or Full (use only to eliminate dupes)
XrefType=Numeric
```

When XrefType=Numeric, only the ObjectID is included in the reference, rather than the full text of the referenced source.

When XrefType=Full, **Mif2Go** includes the full text of cross references in RTF bookmarks, *which breaks the cross references*. The only time you might need this setting would be if your document contains two cross references with the same starting number (followed by other differences) in the two markers; this is even less likely than finding duplicate ObjectIDs in a document. The preferred remedy is to delete and recreate the less used of the two markers, then resolve all references to that marker. Use XrefType=Full only as a last resort, if both markers have thousands of references and resolving them is too onerous. The result will be inactive cross references in Word.

6.11.2.5 Omitting cross references from RTF output

To actually omit cross references from RTF output, you must specify the following property for each cross-reference format you want omitted:

```
[XrefStyles]
; xref format name = properties (Delete or Text),
; if omitted treated as link
XrefFormat=Delete
```

See also §6.2.5 [Constraining the number of bookmarks in Word](#) on page 148.

See also §6.11.5 [Enabling interfile cross references and hypertext links](#) on page 179.

6.11.3 Converting hypertext links to Word

By default, **Mif2Go** creates active links in Word from hotspots with hypertext markers in your FrameMaker document:

```
[WordOptions]
; UseHyperlinks = Yes (default) or No (ignore all hyperlinks)
UseHyperlinks=Yes
```

When UseHyperlinks=Yes, hotspots with hypertext markers become active, clickable links in Word.

When UseHyperlinks=No, hypertext hotspots are not clickable in Word. Nor are contents or index entries clickable; see §6.12 [Converting generated files to print RTF](#) on page 181.

Problems can arise when a hypertext hotspot includes other markers. **Mif2Go** handles some of these problems; you might have to deal with others:

[Alert markers with tilde are ignored](#)

[Index marker ObjectIDs are excluded](#)

[Ignore selected marker types](#)

[Ignore individual markers](#)

Alert markers with tilde are ignored

Mif2Go ignores hypertext **Alert** markers if the marker content starts with a tilde (~), as used by MicroType TimeSavers.

Index marker ObjectIDs are excluded

Unless you are converting the FrameMaker index, by default **Mif2Go** turns off FrameMaker **Index** marker ObjectIDs. To restore use of ObjectIDs for **Index** markers:

```
[WordOptions]
; KeepIXMarkerIDs = No (default, keep only if [Setup]UseFrameIX) or
; Yes (always keep Unique ObjectIDs for Index markers)
KeepIXMarkerIDs=Yes
```

Ignore selected marker types

To have **Mif2Go** ignore markers of a particular type, you can remap such marker types. For example:

```
[Markers]
alert=Delete
openObjectId=Delete
```

See §29.3 [Remapping marker types and hypertext commands](#) on page 836.

Ignore individual markers

To have **Mif2Go** ignore an individual hypertext marker, you must make the marker conditional, and hide its condition before converting your document. You can do this with a conversion template; see §2.4 [Importing formats from a conversion template](#) on page 67.

6.11.4 Locking hypertext links to allow revision tracking

You can lock hypertext links:

```
[WordOptions]
; LockHyper = No (default, allow edit) or Yes (when revision tracking)
LockHyper=Yes
```

When LockHyper=No, hypertext links are active in Word, and they are updatable in Word. On the other hand, every hypertext link is marked with a change bar in Word when revision tracking is on; see §6.16 [Turning on revision tracking in Word](#) on page 194.

When LockHyper=Yes, hypertext links are neither active nor updatable in Word; however, at least in some versions of Word, locking them might avoid having every link marked with a change bar when revision tracking is on.

6.11.5 Enabling interfile cross references and hypertext links

In this section:

- §6.11.5.1 [Identifying Word link destinations with FileIDs](#) on page 179
- §6.11.5.2 [Creating Word bookmarks for interfile cross references](#) on page 179
- §6.11.5.3 [Making page numbers in interfile links updatable](#) on page 180
- §6.11.5.4 [Mapping file names, extension, or location](#) on page 180
- §6.11.5.5 [Using hypertext “message” commands for external links](#) on page 181

6.11.5.1 Identifying Word link destinations with FileIDs

For active interfile links in Word, link code must include file identification. The following default setting ensures that cross references and hypertext links in your document include FileIDs:

```
[WordOptions]
; UseFileIDs = Yes (default, needed for identifying xrefs) or No
UseFileIDs=Yes
```

When UseFileIDs=Yes, **Mif2Go** includes a FileID in the link code, so links do not get confused if a cross-reference number or ObjectID is not unique. **Mif2Go** assigns FileIDs to your FrameMaker files; see §5.3.4 [Working with Mif2Go FileIDs](#) on page 119.

*Keeping or
replacing FileIDs*

Suppose you are still using an RTF conversion project that has FileIDs listed under [FileIDs] in your configuration file; you can either continue using the existing FileIDs, or switch to the newer (preferred) method.

To continue using FileIDs listed in your configuration file, specify the following option:

```
[Setup]
; UseLocalFileID = No (default, use IDFile IDs)
; or Yes (use [FileIDs] here)
UseLocalFileID=Yes
```

See §5.3.4.4 [Keeping legacy FileIDs in the configuration file](#) on page 122.

To switch to mif2go.ini FileIDs, stay with the default, UseLocalFileID=No; and update settings in any configuration sections that reference the original FileIDs, such as the [Graph*] sections. See §5.3.4.3 [Updating files and references when FileIDs change](#) on page 121.

6.11.5.2 Creating Word bookmarks for interfile cross references

By default, **Mif2Go** reproduces FrameMaker cross references between files in RTF output:

```
[WordOptions]
; ExternalXrefs = Yes (default, create all of the bookmarks for each
; para that has an xref marker in it, for links from ext files) or No
; (create xref bookmarks only as required by refs in the current file)
ExternalXrefs=Yes
```

When ExternalXrefs=Yes, **Mif2Go** creates bookmarks around all possible cross-reference sources (<\$paratext>, <\$paranum>, and <\$paranumonly>) in all paragraphs that contain cross-reference markers, because there is no way to determine which sources and which markers are actually referenced. In Word you can INCLUDE and specify a bookmark, and get exactly the same effect as with an interfile REF system. And, if Word cannot find the referenced file, no error message appears.

When ExternalXrefs=No, **Mif2Go** creates bookmarks only for cross-reference markers that are referenced from within the same file. You could set ExternalXrefs=No to

decrease file size. However, this is useful only if your FrameMaker document has both of the following characteristics:

- *many* paragraphs with cross-reference markers
- *no* external files that reference those markers.

6.11.5.3 Making page numbers in interfile links updatable

By default, **Mif2Go** makes page numbers in external references updatable:

```
[WordOptions]
; ExtXrefPages = Yes (default, make page refs to external files into
; updatable fields), or No (just put page numbers as plain text)
ExtXrefPages=Yes
```

When ExtXrefPages=Yes, external cross references with page numbers get the page number from the external file. However, if that file has not itself had its fields locked or updated (see §6.11.2.2 [Making cross references active and updatable](#) on page 175), the page-number value might not be correct. Once you update the page-number field in the referenced file, and then in the referencing file, you should get the right page number in the cross reference. Or, make sure LockXrefs=Yes (the default setting).

6.11.5.4 Mapping file names, extension, or location

In most situations, interfile cross references and hypertext links function as active, updatable links in Word with just the default settings for cross references and hypertext links, provided your RTF output file names have both of the following characteristics:

- extension `.rtf` (see §6.2.3 [Specifying output file extension](#) on page 147)
- the same base names as the corresponding FrameMaker input files.

However, in the following situations you must specify additional settings to make interfile references work:

- You specified an extension other than `.rtf` for output files (see §6.2.3 [Specifying output file extension](#) on page 147).
- You specified names for output files that differ from names of the corresponding input files (see §37.6 [Specifying output file paths and names](#) on page 1002).
- After conversion, the file names or extension will change in a post-processing step; for example, someone will load the output files in Word and then save them for further use with an extension other than `.rtf`.

For interfile cross references and hypertext links you can specify the following:

[Different file extension](#)

[Different file names](#)

[Default file location](#)

*Different file
extension*

To specify the file extension to use in cross references and hypertext links:

```
[WordOptions]
; XrefFileSuffix = suffix to use to convert [WordXrefFiles] xrefs
XrefFileSuffix=ext
```

This setting specifies the extension to use in Word INCLUDE fields and HYPERLINK fields when these fields reference a file other than the current file.

If you change the output file extension in a post-**Mif2Go** step, for example by saving the files as `.doc` from within Word, specify the final extension you expect the files to have at the time the cross references and hypertext links are used.

*Different file
names*

To map input-file base names to corresponding output-file base names:

```
[WordXrefFiles]
; file name in xref = file name for Word interfile xref
fchap1=wchap1
```

List mappings in the form *oldfile=newfile*, without file extension.

*Default file
location*

For some versions of Word, you might have to specify the default document location to be the path to the directory where your converted files are located. From the top menu bar in Word, go to: **Tools > Options... > File Locations > Documents**.

6.11.5.5 Using hypertext “message” commands for external links

Mif2Go produces links to external sources from hypertext markers in your document that contain the following types of hypertext “message” commands:

message URL

message openfile

When a **message openfile** link specifies an absolute path (which must start with a drive letter), **Mif2Go** removes any “file:///” URL prefix to the path, which is not needed in RTF. For example:

```
message openfile file:///g:/omnisys/ug/out/ugmif2go.pdf
```

becomes (in Word 2003):

```
{\*\fldinst {HYPERLINK {g:/omnisys/ug/out/ugmif2go.pdf}}\n }
```

6.11.6 Replacing building blocks in master-page references

Word does not have equivalents to `<$paranum>` and `<$paranumonly>`. When these building blocks are used in cross references to text that appears on FrameMaker master pages (such as in references to page numbers), by default **Mif2Go** substitutes a # character. You can specify the correct number (or other text sequence) based on the name of the FrameMaker file being converted:

```
[WordSectionFiles]
; file name = text for <$paranum> or <$paranumonly>, default "#"
chap1=1
intro=2
config=3
```

Mif2Go assumes that a single FrameMaker file will produce a single Word section, at least with respect to page numbering. It does not matter if there are really ten Word sections for a FrameMaker chapter; all will have the same chapter number. This is like the `<$chapnum>` building block in FrameMaker 6+, specified for each FrameMaker file.

6.12 Converting generated files to print RTF

In this section:

- §6.12.1 [Specifying which generated files to convert](#) on page 182
- §6.12.2 [Activating links in converted index and list files](#) on page 182
- §6.12.3 [Making the entire text of each list entry an active link](#) on page 182
- §6.12.4 [Ensuring link targets are present in RTF output](#) on page 183
- §6.12.5 [Correcting <\\$nopcode> index links](#) on page 184

See also:

- §6.17.2 [Including index terms in Word](#) on page 195

6.12.1 Specifying which generated files to convert

To have **Mif2Go** convert FrameMaker TOC, IX, and other generated files, do one of the following:

- Check appropriate file options in the *Set Up* dialog; see §6.2.2 [Choosing set-up options for a print RTF project](#) on page 146.
- Include appropriate settings in the configuration file; see §5.5 [Converting FrameMaker-generated files](#) on page 124.

*Ensuring
accuracy of page
numbers*

When you convert a FrameMaker-generated file to Word, such as a TOC or IX, **Mif2Go** does not recompute page numbers. If you want those generated files to be useful in Word, you might have to shrink the contents; see §6.5.4 [Maintaining pagination in Word](#) on page 152. Or, you can make the index page numbers into active links; see §6.12.2 [Activating links in converted index and list files](#) on page 182.

6.12.2 Activating links in converted index and list files

For links in FrameMaker-generated files to become active links in Word, the following default value must be in effect:

```
[WordOptions]
; UseHyperlinks = Yes (default) or No (ignore all hyperlinks)
UseHyperlinks=Yes
```

When UseHyperlinks=Yes, links in FrameMaker-generated files become active, clickable links in Word. See §6.11.3 [Converting hypertext links to Word](#) on page 178.

When UseHyperlinks=No, links in FrameMaker-generated files are converted to plain text.

Index files

To get active links for multiple-page entries in output from FrameMaker IX files, you must apply a character format to the page numbers; without the character format, multiple page references for an index entry might cause a crash when UseHyperlinks=Yes. See §5.5.3 [Activating hypertext links in a converted index](#) on page 125.

List files

Because links in generated list files are by nature external to the files they reference, you might have to provide additional settings to identify the files involved. See §6.11.5 [Enabling interfile cross references and hypertext links](#) on page 179 and §6.12.4 [Ensuring link targets are present in RTF output](#) on page 183.

6.12.3 Making the entire text of each list entry an active link

In FrameMaker-generated list files, entries that include a format change have their associated hypertext links truncated at the point of change. For example, if an entry includes a word to which a character format has been applied, the only part of the entry that is an active link would be the text that precedes the character-formatted word.

To make the entire text of every entry an active link regardless of character format changes, assign property ParaLink to the paragraph format. For example:

```
[WordStyles]
; paragraph format = ParaLink
; ParaLink prevents any char formats in the named para format from
; affecting the hotspot area for a link in that para.
*TOC = ParaLink
*LOF = ParaLink
*LOT = ParaLink
```

See also:

§5.10.2 [Making an entire paragraph into a hotspot](#) on page 138

6.12.4 Ensuring link targets are present in RTF output

If the target of a link from a FrameMaker-generated list (such as TOC, LOF, or LOT) is not referenced from any other point in your document, the ObjectID for that target might not be present in Word output. This is because the default is to retain only referenced IDs (see §5.3 [Identifying files and objects](#) on page 117). **Mif2Go** cannot tell from a file itself whether any ObjectIDs are being referenced from other files.

Note: This is a problem only for converted *lists*; targets of links from converted *indexes* are always present in RTF output.

To ensure that converted-list link targets are present, you have a couple of options:

[Include all ObjectIDs](#)

[Assign a format property.](#)

The following method is deprecated:

[Assign a “level” number.](#)

*Include all
ObjectIDs*

To include all ObjectIDs in RTF output:

```
[WordOptions]
ObjectIDs=All
```

Including all ObjectIDs can increase the size of each output file, by providing a Word bookmark for every item. However, this setting does guarantee that links can be completed. See §6.11.1 [Including ObjectIDs for Word links and cross references](#) on page 175.

*Assign a format
property*

To assign an ObjectID-creating format property to each referenced paragraph format (for example):

```
[WordStyles]
; KeepID retains the Frame ObjectID, so hyperlinks in Frame-generated
; files (TOC, LO*, IO*) work in Word; not needed for IX files.
ChapTitle=KeepID
Head1=KeepID
Head2=KeepID
Figure=KeepID
Table=KeepID
```

These settings cause **Mif2Go** to retain paragraph ObjectIDs for the formats listed. (The same is true for the deprecated [WordStyles]Contents property.)

Note: Sometimes character formats can interfere with paragraph ObjectIDs. If any target paragraphs have character formats applied, you might have to use the first method instead: [Include all ObjectIDs](#).

*Assign a “level”
number*

This method is deprecated; use one of the others instead. To assign an ObjectID-creating “level” number to referenced paragraph formats (for example):

```
[WordCntStyles]
; style = 0 (the default), or 1 to create link targets
Figure=1
Table=1
```

The integer after the equals sign can be any integer greater than zero; the actual value does not matter. Assigning a non-zero integer causes **Mif2Go** to retain paragraph ObjectIDs for the formats listed.

6.12.5 Correcting <\$nopage> index links

A FrameMaker-generated index includes, for every <\$nopage> entry, a link to the original marker location in the document. The Sundorne Communications IndexRef plug-in can change the links for <\$nopage> *See* and *See also* entries in FrameMaker, so the links point to the referenced index entries instead. See:

<http://www.sundorne.com/FrameMaker/IndexRef/indexref.htm>

When you convert a generated index to Word, **Mif2Go** converts any IndexRef-created links properly. If you do not use IndexRef, links for *See* and *See also* entries in Word point to the original location of the <\$nopage> index markers in your document.

6.13 Converting tables to print RTF

In Word, tables are not objects. They are just paragraphs with border properties that make them look like rows. Therefore, converted tables might not look quite the same in Word as they do in FrameMaker. For example, titles of tables that span all columns and sideheads in FrameMaker might not line up with the left edge of the table in Word. And multiple table header rows might be split across page boundaries.

You might have to use a conversion template (see §30.7 [Applying FrameMaker conversion templates](#) on page 863) to reduce font sizes in table cells, and reduce cell margins, by 5% to 10%, to keep pagination the same as in FrameMaker; see §6.5.4 [Maintaining pagination in Word](#) on page 152.

To prevent table titles from being separated from their tables across Word page boundaries, and to keep header rows together, in FrameMaker set **Keep With: Next Pgf** for all table-title and table-header paragraph formats.

You can use configuration settings to control some table characteristics for RTF output:

- [Table title](#)
- [Graphics in tables](#)
- [Table indents](#)
- [Cell properties](#)
- [Straddled columns](#)

For other differences you might have to supply a conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67), or modify table formats in your FrameMaker document:

- [Space after tables](#)
- [Rotated cell content](#)
- [Straddled rows](#)
- [Landscape tables](#)
- [Tables with too many columns](#)

If you use conditional text to hide all the rows in a table (but not the table anchor), you will see extra space in the RTF output where the table would have been displayed. To avoid this space, be sure to hide the table anchor.

Table title

To reposition table titles, and specify whether to remove table variables:

```
[Tables]
; TableTitles = 0 to leave alone, 1 to put at top, 2 to put at bottom
TableTitles=0
; TableContinued = No (default) to remove variable from table titles
TableContinued=No
```

```

; TableContVar = name of the variable used for table (continued)
TableContVar=Table Continuation
; TableSheet = No (default) to remove this variable from table titles
TableSheet=No
; TableSheetVar = name of the variable used for table (Sheet m of n)
TableSheetVar=Table Sheet

```

In Word, table titles are not always positioned the same horizontally as in FrameMaker. To try to force table titles to maintain FrameMaker positioning in Word (this does not always work):

```

[Tables]
; TitleInRow = No (default), or Yes (puts table title in a row sized
; according to Frame's implicit rules for table title boxes)
TitleInRow=Yes

```

Graphics in tables To reposition graphics in table cells:

```

[Tables]
; TableGraphics = Standard (default, in cell), None, or Outside
; applies only to non-inline and non-runin frames anchored in cell
TableGraphics=Standard

```

FrameMaker allows a graphic to overflow its table cell, and overlap cells to the right; Word does not. If your document has tables that contain graphics, make sure any cells under the graphics are straddled.

Table indents To remove table indents:

```

[Tables]
; ShiftWideTablesLeft = Yes (default, unindent overwidth tables) or No
ShiftWideTablesLeft=Yes

```

Cell properties To adjust cell properties for all tables:

```

[Tables]
; TableRules = Cell (standard default), None (help default), or one
; of the Box types: Box, Double, Thick, Shadow, Para (variable)
TableRules=Cell
; TableFill = AsIs (default), ColorOnly, ShadingOnly, None
TableFill=AsIs

```

Cell properties set in [Tables] apply to all tables in your document.

If TableRules has any value except Cell, **Mif2Go** does not write borders. If TableFill=None or ColorOnly, **Mif2Go** ignores the table-format configuration settings for shading.

Space after tables If your FrameMaker table format includes a space-below value greater than zero, **Mif2Go** adds a blank “spacer” paragraph after the table in the RTF output, because Word has no way to represent the space-below feature of a FrameMaker table.

Rotated cell content Word 9/2000 and earlier versions cannot handle rotated table cells, unless you put the content of each cell in an anchored frame and rotate it inside the frame. The text will appear correctly in Word unless you try to edit the picture created from the anchored frame; then Word remembers that it cannot rotate text, and unrotates everything. Word 10/XP does allow rotated text in selected table cells, but does not allow rotation of a whole table.

Straddled columns Given the opportunity, Word handles table cells that straddle columns by combining the cells involved in the straddle into a single cell. Because this might not be what you want in Word output, by default **Mif2Go** does *not* combine the cells; however, you can override the default. To combine column-straddling cells into a single cell:


```
[Tables]
; MergeStradCells = No (default) or Yes (combine col-straddling cells)
MergeStradCells=Yes
```

Note: For WinHelp output, the default value of MergeStradCells is Yes (the opposite of the default for Word); see §8.5.2 [Adjusting table appearance](#) on page 261.

<i>Straddled rows</i>	Word 8/97 and later versions can handle straddled rows in tables; Word 7/95 cannot.
<i>Landscape tables</i>	Word never allows a landscape table on a portrait page, with headers and footers in normal portrait positions. If you have a landscape page, you get headers and footers running the long way.
<i>Tables with too many columns</i>	Word seems incapable of handling tables that have more than 63 columns. Such a table ends up in Word with all columns beyond the 63rd merged into the last column allowed, making that cell much taller than the rest, in every row. As a workaround, you can save the FrameMaker file that contains the table as plain text, with tab delimiters for the table cells and hard returns for the rows. Discard everything in the resulting .txt file except the table, and import the file into Excel. Select all columns, resize their widths if necessary, and print to PDF, reducing the size if necessary. Then import the PDF into Word.

6.14 Managing graphics for print RTF

In this section:

- §6.14.1 [Understanding graphics requirements for Word](#) on page 186
- §6.14.2 [Converting referenced graphics](#) on page 187
- §6.14.3 [Converting embedded graphics](#) on page 189
- §6.14.4 [Limiting bitmap resolution and color depth](#) on page 190
- §6.14.5 [Managing callouts added to graphics](#) on page 190
- §6.14.6 [Positioning graphics and wrapping text](#) on page 191
- §6.14.7 [Preserving graphics scale in Word](#) on page 191
- §6.14.8 [Accommodating graphics in multiple versions of Word](#) on page 192
- §6.14.9 [Including file names of referenced graphics in Word](#) on page 192
- §6.14.10 [Linking instead of embedding referenced graphics](#) on page 193
- §6.14.11 [Embedding graphics in converted RTF files](#) on page 193
- §6.14.12 [Updating fields in Word to show linked graphics](#) on page 193

See also:

- §5.7 [Processing graphics](#) on page 126
- §31 [Working with graphics](#) on page 869

6.14.1 Understanding graphics requirements for Word

To produce properly scaled images in Word at the best resolution, you need BMP (for bitmap) or WMF (for vector) versions of the images in your FrameMaker document. Word allows images to be scaled (to match their sizes in FrameMaker) only if the images are embedded as WMFs. Only WMF or BMP images can be embedded as WMFs in Word. **Mif2Go** wraps BMPs in WMFs for scaling purposes.

This is the only way to set the image scale and position in RTF.

Therefore, even though you can insert other types of graphics into a Word document, images to be transferred from your FrameMaker document (or substituted for the images in your FrameMaker document) must be in either BMP or WMF format. Also, Word

requires a viewable WMF of each image to be present in the document file, even if you “link” to the image.

Because Microsoft tinkers with the poorly documented WMF format, if your graphics do not embed correctly, you might have to make do with referencing them in Word via INCLUDEPICTURE field. See §6.14.10 [Linking instead of embedding referenced graphics](#) on page 193

Mif2Go processes directly the following types of images in your FrameMaker document:

- referenced or embedded WMF or BMP images that are alone in their anchored frames (without callouts or other added elements)
- embedded FrameImage graphics
- vector images created with FrameMaker drawing tools.

Other images that are *not* WMF or BMP must be converted; the method depends on how the images are included in your FrameMaker document:

[Referenced graphics](#)

[Embedded graphics](#)

Or, you can supply WMF or BMP equivalents for those images; see §31.3.2.1 [Substituting graphics files for RTF](#) on page 890.

Use WMF only for vector images

The bitmap part of each image must be in BMP format; use WMF only for real vector graphics, not for bitmaps. Converting bitmap graphics to WMF increases processing time dramatically, because **Mif2Go** has to take each such WMF graphic apart and adjust many settings before embedding the image.

Referenced graphics

If your FrameMaker document references graphics that are neither BMP nor WMF, you must provide BMP or WMF replacements for the graphics files; see §6.14.2 [Converting referenced graphics](#) on page 187.

For referenced graphics in other formats, **Mif2Go** puts each image file name in a Word INCLUDEPICTURE field. What Word does with these images depends on which version of Word is used to view the images. Because Word provides no way to specify scaling in an INCLUDEPICTURE field, images in formats other than BMP or WMF are seldom the proper size, unless they were included in FrameMaker at 96 DPI; see §6.14.2.3 [Leaving graphics unconverted for Word \(no scaling\)](#) on page 188, and §6.14.12 [Updating fields in Word to show linked graphics](#) on page 193.

Embedded graphics

If your FrameMaker document includes embedded graphics that are neither BMP nor WMF, by default **Mif2Go** exports those graphics and saves them as external graphics files. See §6.14.3 [Converting embedded graphics](#) on page 189.

6.14.2 Converting referenced graphics

If your document references graphics that are not in BMP or WMF format, those graphics must be converted to BMPs, to embed the images in Word. They do not have to be processed every time you run the conversion, but only when graphics change.

In this section:

§6.14.2.1 [Converting referenced graphics before Mif2Go \(best quality\)](#) on page 188

§6.14.2.2 [Converting referenced graphics with Mif2Go \(least effort\)](#) on page 188

§6.14.2.3 [Leaving graphics unconverted for Word \(no scaling\)](#) on page 188

See also:

§6.14.10 [Linking instead of embedding referenced graphics](#) on page 193

6.14.2.1 Converting referenced graphics before Mif2Go (best quality)

If you want properly scaled images in Word at the best resolution, use a third-party graphics converter to make matching BMP images for referenced graphics that are in other formats. This gives the best quality.

Note: This is your *only* choice if you are running **Mif2Go** via command line.

Although **Mif2Go** can create BMP images from other formats, that process uses the FrameMaker graphic export filters, which are limited to screen resolution. This is acceptable for documents viewed only on screen, but the images look fuzzy when printed. Preconverting preserves the original resolution. It also allows **Mif2Go** to include any callouts or other FrameMaker additions as vector elements in the WMF wrapper.

*Replacements in
project directory*

Put the resulting .bmp files in the project directory, and set the following options in the configuration file:

```
[Graphics]
FilePaths=None
FileNames=Map
```

*Replacements in
original directory*

As an alternative, put the resulting .bmp files in the same directory with the original graphics, and use the following options instead:

```
[Graphics]
FilePaths=Retain
FileNames=Map
```

See §31.3.2.1 [Substituting graphics files for RTF](#) on page 890.

*Map old
extension to new*

For each of the formats you are converting, specify the before and after formats. For example:

```
[GraphFiles]
eps=bmp
gif=bmp
```

See §5.7 [Processing graphics](#) on page 126.

6.14.2.2 Converting referenced graphics with Mif2Go (least effort)

You can have **Mif2Go** convert referenced graphics using FrameMaker graphic export filters. You get screen resolution and rather poor quality (especially for EPS; see §31.2.2.3 [Converting EPS graphics](#) on page 875), but this is the easiest choice.

Note: For graphics that are not alone in their anchored frames, this is the *only* choice.

In the *Export* dialog, choose **Write for anchored frames**. **Mif2Go** uses FrameMaker graphic export filters to convert each anchored frame to a WMF, which is subsequently embedded in the Word RTF file. This is a one-click solution, but it results in graphics that are at screen resolution, and therefore might not print as well in Word as those produced by a third-party converter. For example, screenshots will not be as sharp.

See §5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129.

6.14.2.3 Leaving graphics unconverted for Word (no scaling)

If a graphics file imported into (or exported from) your FrameMaker document is in a format other than BMP, WMF, RF (FrameImage), or FrameMaker native vector, and you do not specify how it should be converted or mapped, by default **Mif2Go** plunks the name of the graphics file into the RTF output file and lets Word try to cope. Sometimes it can.

For each such graphic, **Mif2Go** embeds an INCLUDEPICTURE field in the Word file, and leaves it to Word to import the graphic. There is no way to specify scaling. Word imports

the graphic at its “native” size, based on 96 DPI. If you imported the graphics *by reference* into FrameMaker at 96 DPI, the scaling should come out right. See §6.14.12 [Updating fields in Word to show linked graphics](#) on page 193.

Word does not like file paths in INCLUDEPICTURE fields, so you might want to set the following option:

```
[Graphics]
FilePaths=None
```

See also §31.6 [Converting graphics with Microsoft Word filters](#) on page 904.

6.14.3 Converting embedded graphics

If your FrameMaker document contains embedded graphics that are not in BMP or WMF format, usually those graphics should be converted to BMPs. By default, **Mif2Go** exports such graphics and saves them in your project directory as external graphics files, in their original formats, each with a name that starts with the name of the FrameMaker file from which it was exported, and ends with a string of digits (see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134).

After you have run the conversion and the automatically exported graphics are in your project directory, you have pretty much the same choices as for referenced graphics; see §6.14.2 [Converting referenced graphics](#) on page 187.

In this section:

§6.14.3.1 [Converting exported graphics outside of Mif2Go](#) on page 189

§6.14.3.2 [Using exported graphics in their original format](#) on page 190

See also:

§31.2.3 [Exporting and converting embedded graphics](#) on page 877

6.14.3.1 Converting exported graphics outside of Mif2Go

You can convert automatically exported graphics files to BMPs with a graphics program; see §5.7.2.3 [Using third-party graphics converters](#) on page 130. Give the converted files the same file names as the exported graphics, but with extension `.bmp`, and leave them in the project directory. Then, in your configuration file specify the following options:

```
[Graphics]
FileNames=Map
FilePaths=None

[GraphFiles]
oldext=bmp
```

For example, if the original graphics are in JPEG format, you would substitute `jpg` for `oldext`:

```
[GraphFiles]
jpg=bmp
```

Now rerun the conversion. This time **Mif2Go** will use the `.bmp` files instead of their embedded counterparts, and will include the BMP graphics in your `.rtf` output.

See also:

§5.7.3.2 [Processing embedded graphics separately](#) on page 132

6.14.3.2 Using exported graphics in their original format

If the original graphics embedded in your document are photographs in JPEG format, converting to BMP degrades the quality of the image. You might consider leaving the exported graphics in their original formats.

Automatically exported graphics will be referenced in RTF provided you load the graphics in Word while the graphics files are still in the project directory. However, the scaling used in FrameMaker is not preserved, so you might have to resize each image in Word. The advantage is better image quality; the downside is the manual work.

See also:

§6.14.2.3 [Leaving graphics unconverted for Word \(no scaling\)](#) on page 188

6.14.4 Limiting bitmap resolution and color depth

Graphics destined for print look best at a resolution of 300 to 1,200 DPI. However, you cannot use FrameMaker to increase the DPI of an existing bitmap image: the upper limit of resolution is the DPI of the original graphic.

Use 256-color (8-bit) BMP images where possible

The number of colors in bitmap images should be 256 or less, because the size increase for more colors can make files too large for Word to load. If you use 24-bit color, conversion can be slower and the resulting .rtf files much larger. This is because **Mif2Go** has to use embedded WMFs to get the images into Word, and 24-bit bitmaps do not compress in WMFs. If you are using 256 colors (8-bit bitmaps), conversion is faster and output file size smaller. However, reducing a photographic image to 256 colors degrades its appearance.

Save Word files with 24-bit BMP images as .doc

RTF lacks compression for 24-bit BMP images (which include “millions of colors”). Every pixel takes 3 bytes. At 300 DPI for acceptable print quality, a 3.25" x 3.5" image would contain 1,023,750 pixels, requiring about 3 MB in a binary format. However, because RTF represents each byte with *two* hexadecimal digits, the actual size would be 6 MB. On the other hand, if you are able to load an RTF file containing such an image into Word, you can save the file as a Word .doc file. Then Microsoft uses a proprietary compression method to shrink the size drastically. Unfortunately, **Mif2Go** cannot legally use this method, thanks to the Digital Millennium Copyright Act (DMCA).

See also:

§31.2.1.2 [Converting bitmap graphics for print RTF](#) on page 873

6.14.5 Managing callouts added to graphics

To keep callout text from reflowing, **Mif2Go** renders as text lines any callouts that consist of FrameMaker paragraphs inside text frames. In some cases this can cause poor results if you use Save as PDF in Word 2007. If the converted files must go from Word to PDF, print to postscript from Word instead of using Save as PDF.

For WMF or BMP graphics, callouts inserted with FrameMaker drawing tools should convert without a problem, unless the callouts involve one or more of the following:

[White text](#)

[Rotated text](#)

[Multiple fonts](#)

White text

If the callouts are in white text, make sure the following default setting is in force:

```
[WordOptions]
HideWhiteText=No
```

- Rotated text* The WMF format used by Word does not support rotated text. You will see the text rotated in the WMF image, but if you try to edit the image, that text will unrotate itself, and you cannot put it back as it was.
- Multiple fonts* If there are font changes within a text line, you might have to adjust character metrics to avoid spacing anomalies at the point of change. See §6.8.4 [Altering font metrics to adjust tabs](#) on page 165.

6.14.6 Positioning graphics and wrapping text

- Specifying a paragraph format for graphics* You might want to override the default positioning of all regular anchored frames (such as screen shots), so you can make them more consistent; for example:

```
[WordOptions]
; FrameStyle = para style for non-in-line anchored frames
; default is not to specify, which uses the previous para style
FrameStyle=Picture
```

The default is not to specify a format, which causes graphics to use the previous paragraph format.

- Specifying text wrapping* Word does not handle text wrapping the same way as FrameMaker. If text is not wrapping properly around a graphic, experiment with the following setting:

```
[WordOptions]
; WrapAroundTextFrames = Yes (default, leave room around) or No
WrapAroundTextFrames=No
```

The default is *Yes*, set for text to wrap.

6.14.7 Preserving graphics scale in Word

Between Word versions 7/95 and 8/97 Microsoft changed the size of a graphics scale unit from *twips* (twentieths of a point: 1,440 per inch) to *himetric* (hundredths of a millimeter: 2,400 per inch). For Word 7/95, **Mif2Go** computes the value in twips; for Word 8/95 and later versions, in himetric; until Word 2003, when Microsoft changed the (non-user-settable) graphics scale unit back to twips.

Table 6-6 shows what happens to the apparent size of an embedded image viewed in Word, depending on which version of Word you specify as the **Mif2Go** output type, and which version of Word you use to view the RTF output. The rightmost column shows the settings required to preserve scale for those versions of Word that display images at other than 100%. These settings affect only images embedded in Word. Linked images cannot be scaled; see §6.14.10 [Linking instead of embedding referenced graphics](#) on page 193.

Table 6-6 Graphics scale percentages for Word versions

Project output type	Image scale for RTF viewed in Word, by version					[WordOptions] remedial setting to achieve 100%
	7/95	8/97	9/2000	10/XP	11/2003+	
Word 7/95	100%	60%	60%	60%	100%	Word8 = Yes
Word 8/97	167%	100%	100%	100%	167%	Word2003 = Yes

The remedy depends on which problem you observe in Word:

- [Embedded images are much too small](#)
- [Embedded images are much too large](#)
- [Embedded images are still a little off](#)

*Embedded
images are much
too small*

If you specify **Word 7/95** as the output type when you set up your conversion project, then convert your document and load the resulting RTF file(s) in Word 8/97, Word 9/2000, or Word 10/XP, your graphics will appear at 60% of the correct size. If you are converting from within FrameMaker, the remedy is to change the output type in the *Choose Project* dialog (see §3.3 [Creating a Mif2Go conversion project](#) on page 78) to **Word 8/97**, and click **Modify**. To make the change persist, also specify this setting in the configuration file:

```
[WordOptions]
Word8 = Yes
```

*Embedded
images are much
too large*

If you specify **Word 8/97** as the output type when you set up your conversion project, then convert and load the output into Word 7/95 or Word 11/2003, your graphics will appear at 167% of their original size. To correct this problem, add the following setting:

```
[WordOptions]
Word2003 = Yes
```

*Embedded
images are still a
little off*

For Word versions that use a default graphic unit of “himetric” instead of “twips” (Word 8/97, 2000, and 10/XP), the correctly computed scaling factor of 176 does not always look right. To adjust the scaling factor:

```
[WordOptions]
; PicScale = 176 (default), percentage to expand graphics for Word
; 8/97, 9/2000, and 2002/XP to compensate for redefined Word default.
PicScale = 176
```

Adjust as needed; for example, Word 9/2000 seems to do better with PicScale=178.

6.14.8 Accommodating graphics in multiple versions of Word

If your converted files are to be used in a later version of Word (or in several versions), and you expect problems with missing or incorrectly scaled graphic elements, do the following for each RTF file:

1. Load the file into the version of Word you specified for the conversion.
2. Save the file as .doc.

Use the .doc file instead of the RTF file in later versions of Word.

See also:

§6.17 [Managing Word output after conversion](#) on page 195

§6.3 [Adjusting output for different versions of Word](#) on page 149.

§6.14.10 [Linking instead of embedding referenced graphics](#) on page 193.

6.14.9 Including file names of referenced graphics in Word

*Name and
graphic*

To include the original file names of FrameMaker referenced graphics in Word output, so reviewers can refer to graphics by name:

```
[Graphics]
; NameGraphics = No (default)
; or Yes (for Word only, put original Frame graphic name in an
; INCLUDEPICTURE field, with the WMF in the result part of the field,
; so that the name is shown by showing Field Codes)
NameGraphics=Yes
```

When NameGraphics=Yes, **Mif2Go** inserts a Word field that has the following content:

- the file name of the graphic in the INCLUDEPICTURE field instructions
- the corresponding WMF in the field result part.

The entire field is locked so it cannot be updated accidentally. Reviewers can view the names of graphics in the Word document via **Tools > Options > View > Field Codes**.

Name only To include and display graphics file names but omit the graphics themselves, see §31.3.2.5 [Excluding graphics from RTF output](#) on page 895.

6.14.10 Linking instead of embedding referenced graphics

You can use **Mif2Go** to produce RTF output with linked graphics instead of embedded graphics (in fact, that is what you get by default when no BMP or WMF versions of referenced graphics are present). Usually this is not a good idea, because Word does not permit scaling linked graphics in RTF. For an image to appear at the correct size, it must be in an embedded WMF; see §6.14.1 [Understanding graphics requirements for Word](#) on page 186.

To create RTF files without embedding referenced graphics, so that Word can link to the graphics files instead, make sure that **Mif2Go** can neither find nor generate WMF or BMP versions of those graphics. Make sure no referenced graphics are in the project directory. Then **Mif2Go** will create INCLUDEPICTURE references to the graphics instead of embedding them; see §6.14.2.3 [Leaving graphics unconverted for Word \(no scaling\)](#) on page 188.

After converting your document, place the graphics files to be linked in the same directory as the RTF files. To see the graphics in Word, go to **Edit > Links...**, select each graphics link, uncheck **Save picture in document**, and click **Update Now**.

See also:

§6.3 [Adjusting output for different versions of Word](#) on page 149

§6.14.9 [Including file names of referenced graphics in Word](#) on page 192

§6.14.11 [Embedding graphics in converted RTF files](#) on page 193

6.14.11 Embedding graphics in converted RTF files

If your conversion project is set up so that **Mif2Go** inserts field references to graphics instead of the graphics themselves (because they are not in WMF or BMP format, for example), you might want to do the following:

1. Open the `.rtf` file in Word. When you do so, Word imports all the graphics named in the fields, if possible.
2. Save the file as `.doc`. The `.doc` file includes the graphics, and you do not need to provide the graphics files themselves along with the Word file.

If you provide only the `.rtf` file, you have to provide any graphics files also. To find out if the `.rtf` file contains graphics, check the size: files containing graphics are much larger than files that only reference graphics.

See also:

§6.14.9 [Including file names of referenced graphics in Word](#) on page 192

§6.14.10 [Linking instead of embedding referenced graphics](#) on page 193

6.14.12 Updating fields in Word to show linked graphics

When your graphics end up in INCLUDEPICTURE fields, you have to update all fields in Word to show the images; this is a limitation of Word. Use **Ctrl-A** then **F9** in Word to update an entire Word file. If you have a large number of files to do this for, you might want to create a VBA macro in Word to do the updating and then save each file in `.doc` format, so that you do not have to do it again. You should be able to set Word to do the updating on load, as a Word `Auto*` macro.

6.15 Including RTF code for Word output

If you want to do complicated things with RTF, you need to know the coding rules. Microsoft provides very little in the way of documentation, and the specification is incomplete. The best approach is to do what you want in a tiny file in Word itself, save as RTF, and use the Omni Systems pretty-print utility to make the result readable:

```
pprtf myfile.rtf > myfile.txt
```

Study what Word did to produce that output. Make sure the braces {} are balanced in the fragment you extract from the Word output, and double the backslashes.

For example, suppose you want to use a **Mif2Go** macro to pull an image into the title page for your Word document, so in RTF you get an INCLUDEPICTURE field:

```
{ INCLUDEPICTURE "MyLogo.bmp" \* MERGEFORMAT \d \x \y }
```

Because backslash is meaningful both in RTF code and in **Mif2Go** macros, you must double any backslashes within fields and then double them again for the macro; so you end up with something like this to get what you want into RTF:

```
{\field {\*\fldinst INCLUDEPICTURE "MyLogo.bmp" \\\d \\\x \\\y  
\\* MERGEFORMAT }{\fldrslt }}
```

If you look at the resulting RTF with a text editor after you run the **Mif2Go** conversion, you should see:

```
{\field {\*\fldinst INCLUDEPICTURE "MyLogo.bmp" \d \x \y  
\\* MERGEFORMAT }{\fldrslt }}
```

Line breaks are acceptable in RTF code.

6.16 Turning on revision tracking in Word

To turn on revision tracking in Word for **Mif2Go** RTF output files:

```
[WordOptions]  
; RevTrack = No (default) or Yes (turn on Word revision tracking)  
RevTrack=No  
; RevProt = No (default) or Yes (locks on Word revision tracking so  
; that user cannot turn it off, also sets RevTrack=Yes)  
RevProt=No
```

When RevTrack=Yes (or RevProt=Yes), if you leave cross references unlocked (see §6.11.2 [Converting cross references to Word](#) on page 175), you get change bars in Word for every cross reference; and if you also specify live hypertext links (see §6.11.3 [Converting hypertext links to Word](#) on page 178), you get change bars in Word for every hypertext link. You can lock cross references and hypertext links:

```
[WordOptions]  
; LockXrefs = Yes (default, faster load)  
; or No (allow updating of xrefs)  
LockXrefs=Yes  
; LockHyper = No (default, allow edit) or Yes (when revision tracking)  
LockHyper=Yes
```

When you lock cross references, they still work, but you have to unlock them to update them in Word; when you lock hypertext links, the links no longer work in Word.

In some versions of Word, even setting both options does not turn off revision marking of links.

6.17 Managing Word output after conversion

In this section:

- §6.17.1 [Supporting more than one version of Word](#) on page 195
- §6.17.2 [Including index terms in Word](#) on page 195
- §6.17.3 [Producing ASCII text from a converted Word document](#) on page 196
- §6.17.4 [Combining RTF files into a Word master document](#) on page 197
- §6.17.5 [Checking print RTF output files for Mif2Go version](#) on page 197

6.17.1 Supporting more than one version of Word

If you are trying to support multiple users who have a variety of Word versions, you cannot just give them RTF files; Microsoft made sure of that. Instead, you must provide documents in .doc or .docx format.

To produce Word files in .doc or .docx format from RTF files generated by **Mif2Go**:

1. Load each generated RTF file into the version of Word for which **Mif2Go** produced it; see §6.3 [Adjusting output for different versions of Word](#) on page 149).
2. Wait until Word has counted up pages to the end, and stops; watch the counter in the status bar.
3. Select all (**Ctrl+A**).
4. Update fields (**F9**).
5. Save the file *from Word* as .doc (or .docx for Word 2007 and later versions).

Automate the process

Or, you can create a Word VBA macro that will do all that for you. In **Mif2Go** you would use a SystemEndCommand (see §34.4.1 [Specifying system commands](#) on page 938) to open RTF files in Word and invoke a Word macro like the following macro for .doc output:

```
Sub LoadRtfFile()
'
' LoadRtf2007 Macro
' Macro recorded 5/31/2010 by Omni Systems
'

    Dim nameStr As String
    Selection.WholeStory
    Selection.Fields.Update
    nameStr = Replace(ActiveDocument.FullName, ".rtf", ".doc")
    ActiveDocument.SaveAs FileFormat:=wdFormatDocument, _
        FileName:=nameStr, LockComments:=False
    Selection.StartOf
End Sub
```

See the Word VBA reference for details. Word files in .doc or .docx format can be distributed to users who have any version of Word.

The value of SystemEndCommand would look like this:

```
path/to/winword.exe /mLoadRtfFile path/to/<$$_basename>.rtf
```

6.17.2 Including index terms in Word

To have **Mif2Go** include index terms in RTF output:

```
[WordOptions]
; Index = Standard (Word index markers), or None
Index=Standard
```

When Index=Standard, **Mif2Go** creates a Word {xe} field for each index marker. This happens whether or not you also convert a FrameMaker index, which does not use {xe} fields for index references.

Note: Word uses the character formatting in effect at the {xe} field itself for the entire index item. To avoid unwanted formatting, place index markers at the very end of a paragraph. (Or at the very beginning; though if the paragraph format itself includes bold or italic, those effects will apply to the index item.)

When Index=None, Word {xe} fields are not created.

“See also” index references

See also” type index references are not supported by Word index generation, and Word has no concept of omitting page numbers from the indexes it produces. **Mif2Go** cannot prevent the bogus page number in a <\$nopage>see-also index link from appearing in a Word-generated index, because Word has no field switch for that purpose. However, when you specify Index=Standard you can direct **Mif2Go** to refrain from converting <\$nopage> index markers to Word {xe} fields:

```
[WordOptions]
; NoSeeAlso = No (default, keep See Also markers)
;   or Yes (remove them)
NoSeeAlso=Yes
```

When NoSeeAlso=No, <\$nopage> index markers are converted to {xe} fields in RTF output files. When you generate an index in Word, unless you used IndexRef before converting (see §5.5.4 [Making See and See also index entries into useful links](#) on page 125), the corresponding entries appear with links to the pages where the {xe} fields are located: bogus references.

When NoSeeAlso=Yes, <\$nopage> index markers are not converted to {xe} fields in RTF output.

This option has no effect on indexes that **Mif2Go** converts to Word from the FrameMaker IX file; see §6.12 [Converting generated files to print RTF](#) on page 181.

6.17.3 Producing ASCII text from a converted Word document

If your reason for converting a document from FrameMaker to Word is to take advantage of the Text with Layout converter available from <http://www.gmayor.com/downloads.htm>, you might have to provide some extra settings to cope with differences in how Word treats such things as tabs and cross references.

Replace missing tabs with extra spaces

The Text with Layout converter drops tabs from headings and numbered or bulleted formats. To get around this, for each such format specify one or more fixed spaces to follow the number or bullet. For example:

```
[CodeAfterAnum]
Bulleted = \~\~
Heading* = \~\~
Numbered* = \~\~
```

See §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

Remove unwanted page references

Unwanted numbers might appear at the ends of headings; for example, “Known Issues” might appear as “Known Issues26”. These numbers are hidden-text page numbers that **Mif2Go** uses to emulate dynamic cross references to pages. To omit these numbers:

```
[WordOptions]
ExtXrefPages = No
```

See §6.11.5.3 [Making page numbers in interfile links updatable](#) on page 180.

6.17.4 Combining RTF files into a Word master document

To produce a single RTF file from multiple RTF files:

1. Convert all FrameMaker chapters as usual (from the book, one operation).
2. In Word, create a “master document”.
3. Import each converted RTF file into the master document in Word.

This might be a slow process if you have large chapters, and you might encounter stability problems with Word master documents.

6.17.5 Checking print RTF output files for Mif2Go version

If you recently installed a **Mif2Go** upgrade or beta version, after you run **Mif2Go**, check to make sure the latest version was actually used to produce RTF output. Windows sometimes caches DLLs, and does not always use a newly replaced DLL until after the system is rebooted.

Open an RTF output file in Word and choose **File > Properties > Comments**. You should see a line like the following:

```
DCL filter dwrtf, Ver 3.3 m194b r278b
```

The last two entries identify the build numbers of the **Mif2Go** `drmfif.dll` and `dwrtf.dll` components that were used to create the RTF file. See §D.2.9 [Check your version of Mif2Go](#) on page 1034.

6.18 Converting to OpenOffice or StarOffice

OpenOffice.org Writer and StarOffice can open the RTF files that **Mif2Go** converts from FrameMaker to Word 97 or Word 2000. However, not all features are supported. According to OpenOffice.org 2.0 Help:

OpenOffice.org can automatically open Microsoft Office 97/2000/XP documents. However, some layout features and formatting attributes in more complex Microsoft Office documents are handled differently in OpenOffice.org or are unsupported. As a result, converted files require some degree of manual reformatting. The amount of reformatting that can be expected is proportional to the complexity of the structure and formatting of the source document.

If you load the RTF output files in Word and save them as `.doc` first, then open the `.doc` files in OpenOffice, results are much improved.

7 Producing on-line Help

You can use **Mif2Go** to generate various forms of on-line Help. Special settings are available for Microsoft Windows Help (WinHelp), Microsoft HTML Help, OmniHelp, Oracle Help for Java, and JavaHelp. This section addresses issues that are common to most or all Help systems. Topics covered:

- §7.1 [Weighing Help-system alternatives](#) on page 199
- §7.2 [Setting up a Help system project](#) on page 203
- §7.3 [Producing contents and index for Help systems](#) on page 204
- §7.4 [Configuring contents entries for Help systems](#) on page 209
- §7.5 [Configuring index entries for Help systems](#) on page 211
- §7.6 [Providing related-topic links for Help systems](#) on page 219
- §7.7 [Jumping to secondary windows in Help systems](#) on page 224
- §7.8 [Creating pop-up topics for Help systems](#) on page 225
- §7.9 [Including expandable sections in Help topics](#) on page 226
- §7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239
- §7.11 [Setting up a dynamic modular Help system](#) on page 241

For strategies and configuration settings that are specific to a particular Help system, see the following:

- §8 [Generating WinHelp](#) on page 243
- §9 [Generating Microsoft HTML Help](#) on page 295
- §10 [Generating OmniHelp](#) on page 341
- §11 [Generating JavaHelp or Oracle Help](#) on page 373
- §12 [Generating Eclipse Help](#) on page 403

7.1 Weighing Help-system alternatives

Most users expect three navigation elements in a Help system:

- Table of Contents (TOC), preferably in an expanding and collapsing tree form, that tracks your position in the Help system
- Index (IX), with multiple levels and *See/See Also* capabilities
- Full-Text Search (FTS) that gets you directly to each occurrence of a word or phrase.

You could use generic HTML for a Help system *as is*, especially with framesets (see §13.14 [Using framesets](#) on page 450). JavaScript-based TOC templates are available on the Web, and an index is not hard to create. But the search engine is harder. So it is a good idea to consider the existing alternatives for Help systems. Keep in mind that users might run into security issues with any browser-based help system; all such systems use JavaScript.

In this section:

- §7.1.1 [Considering Help-system features](#) on page 200
- §7.1.2 [Understanding the effects of mid-topic links](#) on page 200
- §7.1.3 [Evaluating Microsoft Windows Help \(WinHelp\)](#) on page 200
- §7.1.4 [Evaluating Microsoft HTML Help](#) on page 201
- §7.1.5 [Evaluating WebHelp](#) on page 201
- §7.1.6 [Evaluating OmniHelp](#) on page 202

§7.1.7 [Evaluating JavaHelp and Oracle Help for Java](#) on page 202

§7.1.8 [Evaluating Eclipse Help](#) on page 202

7.1.1 Considering Help-system features

For HTML-based Help systems, **Mif2Go** can produce any specialized form of HTML you need, including those that work with proprietary DLLs.

If you have the eHelp (RoboHelp) license that permits you to redistribute the appropriate eHelp DLL, you can use **Mif2Go** to produce WinHelp 2000. You generate WinHelp, then add the eHelp data to the WinHelp project file before you compile; see §8.2.12 [Integrating WinHelp from RoboHelp](#) on page 250. Also, you can generate HTML Help as a precursor to using RoboHelp to generate WebHelp; see §7.1.5 [Evaluating WebHelp](#) on page 201.

If you need something similar to WebHelp or Web Works Help, try generating OmniHelp; see §7.1.6 [Evaluating OmniHelp](#) on page 202. Otherwise you would have to roll your own, which you could do with **Mif2Go** frameset support; see §13.14 [Using framesets](#) on page 450. OmniHelp provides a simpler and faster solution.

If you need Microsoft Help Viewer 1.x (the successor to Microsoft Help 2), you can generate HTML Help with **Mif2Go** and then use mshcMigrate (part of the Helpware FAR tool set) to convert the resulting .chm file:

mshcMigrate	http://mshcmigrate.helpmvp.com/home
FAR	http://www.helpware.net/FAR/index.html

Microsoft Help Viewer 1.x is the local Help that ships with Visual Studio 2010 and its associated MSDN Library; see:

<http://www.helpware.net/mshelp3/>

See also §7.5.2 [Preparing index entries for Microsoft Help Viewer](#) on page 211.

To produce electronic books and content for mobile devices, see §13.1 [Deciding which type of output to produce](#) on page 424.

7.1.2 Understanding the effects of mid-topic links

If your FrameMaker document features cross references to items other than titles of topics, be aware of the limitations this level of reference granularity imposes on the Help system.

Several of the HTML-based Help systems are not designed to support large files with many mid-topic links. These Help systems are intended for, and some are optimized for, single-topic files. In OmniHelp, mid-topic links cause problems with navigation; for example, the **Prev/Next** buttons do not work as expected. HTML Help, Eclipse Help, and JavaHelp also have trouble with mid-topic links. In HTML Help, for example, the TOC does not synchronize with the topic being displayed.

7.1.3 Evaluating Microsoft Windows Help (WinHelp)

WinHelp is a very old format, and is not supported on versions of Windows later than Windows XP, and Microsoft Help Workshop is no longer available to compile WinHelp. Your users would have to individually download the WinHelp reader from Microsoft; you are prohibited from redistributing the reader. Many products that originally supported WinHelp dropped it once Microsoft made it effectively impossible to use.

*WinHelp
drawbacks*

Although WinHelp works on all flavors of Microsoft Windows, users must go through a multiple-step validation process to use WinHelp on Windows versions later than Windows

XP. WinHelp does not work on any system other than Microsoft Windows, except through a Windows emulator.

WinHelp does not support mouseovers, and supports Flash movies only with difficulty. Text formatting is limited (especially for tables), you cannot customize index sort order, and there is no tri-pane display.

The WinHelp compiler, which predates Unicode, does not recognize Unicode characters, which instead are in a proprietary Microsoft encoding.

*WinHelp
advantages*

WinHelp provides the fast response needed for Context Sensitive Help (CSH). You can use WinHelp for initial CSH calls, and WinHelp can, in turn, link to HTML Help or OmniHelp for further information. Also, WinHelp produces decent pop-ups.

See §8 [Generating WinHelp](#) on page 243.

7.1.4 Evaluating Microsoft HTML Help

HTML Help from Microsoft does a thorough job, even though it is slow and has numerous defects.

Some disadvantages:

- Your users cannot access compiled HTML Help on a network drive; the CHM file must be local.
- HTML Help does not perform exactly as documented. Some features are missing, others have defects, and the software is no longer being maintained.
- HTML Help requires Internet Explorer 4.x or a later version. HTML Help uses most of the guts of Internet Explorer, which opens the user's system to numerous security hazards via ActiveX features.
- The compressed .chm files can be used only on Windows systems, not on Macintosh or UNIX, because the Java applet is poorly implemented. This is the main reason other Help-authoring-tool vendors use their own proprietary Java applets to provide a tri-pane window and search functionality, which you need for cross-platform applications.
- Pop-ups are just plain text: no font variations appear at all, not even **bold** or *italic*.
- Opening Context Sensitive Help the first time can be very slow.

On Windows 2000, Microsoft itself gets around the last two problems by using WinHelp for Context Sensitive Help and pop-ups, HTML Help for the rest.

7.1.5 Evaluating WebHelp

WebHelp is a proprietary Help format; to generate WebHelp, you must have RoboHelp installed on your system. To produce WebHelp-compatible output with **Mif2Go**, you generate HTML Help, then import the HTML Help project file into RoboHelp.

Generating HTML Help produces contents and index files; when you import the HTML Help project file into RoboHelp, you get the whole contents and index. However, index links in WebHelp can point only to the beginning of a topic. When an indexed item can be displayed at the top of the screen after a jump, users do not have to guess why this particular topic came up for that index entry; this is especially important for topics that contain long tables of values. WebHelp deprives you of that option; see §7.5.8 [Specifying index link destinations for HTML-based Help](#) on page 215.

7.1.6 Evaluating OmniHelp

If you need cross-platform compatibility and easy localization, and you can manage with a JavaScript-based Help system, consider OmniHelp. You can read the OmniHelp Design Report here:

<http://.mif2go.com>

Unlike JavaHelp, Oracle Help, or Eclipse Help, on the client side OmniHelp is based entirely on JavaScript. OmniHelp uses only JavaScript, framesets, and CSS; and therefore works on any operating system, with any current browser. For international use, if you furnish translated versions of text contained in three small JavaScript files, you have a localized interface. A technical writer can edit the text in these files without disturbing the JavaScript code.

OmniHelp provides contents and index, full-text Boolean search with JavaScript-style regular expressions, Context Sensitive Help, related-topic links, pop-ups, and secondary windows.

See §10 [Generating OmniHelp](#) on page 341.

7.1.7 Evaluating JavaHelp and Oracle Help for Java

For pure Java applications, consider Oracle Help for Java.

*Oracle Help for
Java*

Oracle Help for Java, from Oracle, is an excellent choice if the application for which you are preparing Help is written in Java, especially if you need cross-platform compatibility. It is a better alternative to Sun Microsystems JavaHelp. **Mif2Go** writes Oracle Help for Java files in Oracle Help preferred format, rather than just JavaHelp format. You can use most of the features **Mif2Go** supports for JavaHelp. However, you might not be able to create a usable JAR file from an Oracle Help for Java helpset.

JavaHelp

JavaHelp from Sun Microsystems is another choice if your application is written in Java, and if you can tolerate limited CSS support, no support for related-topic linking, only one topic per index entry, and slow performance.

Other Java-based systems can be worth considering, also; see the HelpMaster site for information:

<http://www.helpmaster.info/>

See §11 [Generating JavaHelp or Oracle Help](#) on page 373.

7.1.8 Evaluating Eclipse Help

Eclipse Help is specific to the open-source Eclipse Platform. Eclipse is built on a mechanism for integrating and running modules the Eclipse Community calls *plugins*. An Eclipse plugin connects to an Eclipse Platform at an *extension point*, providing information about itself in an XML *manifest file*. For information about Eclipse, see:

<http://www.eclipse.org/>

Eclipse Help is based on an XML table of contents that specifies the structure of the Help system and references content in standard HTML files. Eclipse Help plugs into an Eclipse Platform, which provides the viewer. Eclipse Help can provide context-sensitive help (in the form of “infopops”) for other Eclipse applications.

Eclipse Help can be challenging to set up, and it is poorly documented as a Help format. It makes most sense if you are documenting an Eclipse plugin, where the environment is already installed on users' systems.

7.2 Setting up a Help system project

In this section:

§7.2.1 [Checking automatic Help topic assignments](#) on page 203

§7.2.2 [Configuring run-in paragraphs](#) on page 203

§7.2.3 [Specifying additional processing after conversion](#) on page 203

§7.2.4 [Compiling and distributing Help systems](#) on page 204

7.2.1 Checking automatic Help topic assignments

When you set up a **Mif2Go** Help project in FrameMaker, **Mif2Go** tries to determine which paragraph formats (usually headings) are most likely to start new topics; see §1.5 [How Mif2Go works](#) on page 62.

Mif2Go uses a heuristic weighting formula to automatically assign topic levels to paragraph formats, taking into account factors such as centering, font size, indents, bold/italic, and so forth; and then ranks the results, grouping close rankings together. Then **Mif2Go** lists, in the following configuration-file sections, the paragraph formats chosen to start new topic files:

<u>Help type</u>	<u>Section</u>
WinHelp	[HelpStyles]
HTML-based Help	[HTMLParaStyles]

Check topic levels!

These topic-level assignments are only a first approximation, and **Mif2Go** is easily fooled. *Be sure to inspect the proposed assignments*, and correct any that are inappropriate. Also check the corresponding contents-level settings; see §7.4 [Configuring contents entries for Help systems](#) on page 209.

7.2.2 Configuring run-in paragraphs

By default, for Help systems **Mif2Go** separates each run-in heading from its following paragraph. To direct **Mif2Go** to emulate FrameMaker run-in headings in Help output:

```
[HTMLOptions] or [HelpOptions]
; RunInHeads = Normal (default for Help systems) or Runin
RunInHeads = Runin
```

When RunInHeads=Normal, **Mif2Go** inserts a carriage return between the run-in heading and the paragraph that follows.

When RunInHeads=Runin, the following paragraph starts on the same line as the run-in heading, as it does in FrameMaker.

7.2.3 Specifying additional processing after conversion

When you set up a new Help project, **Mif2Go** includes a few postprocessing settings in your newly created configuration file.

Some Help systems require running additional programs after files are converted from FrameMaker: either to compile the output (WinHelp or HTML Help), or to create a search index (JavaHelp or Oracle Help). When you first set up a **Mif2Go** project to generate one of these Help systems (see §3.4 [Choosing project set-up options](#) on page 79), you can check an option to have **Mif2Go** run the additional program after conversion.

<i>WinHelp, HTML Help</i>	<p>If you check Compile when you set up a WinHelp or HTML Help project, Mif2Go adds the following settings to your configuration file:</p> <pre>[Automation] CompileHelp = Yes WrapAndShip = Yes</pre> <p>If you do not check Compile, you get the following settings instead:</p> <pre>[Automation] CompileHelp = No WrapAndShip = No</pre>
<i>JavaHelp, Oracle Help</i>	<p>Whether or not you check Make FTS for JavaHelp or Oracle Help, Mif2Go includes the following setting in your new configuration file:</p> <pre>[Automation] WrapAndShip = Yes</pre>
<i>OmniHelp, Eclipse Help</i>	<p>For OmniHelp or Eclipse Help, Mif2Go includes the following setting:</p> <pre>[Automation] WrapAndShip = No</pre>
<i>All output types</i>	<p>Mif2Go also includes the following settings in every new project configuration file:</p> <pre>[Automation] WrapPath = ._wrap ShipPath = .._ship</pre> <p>If CompileHelp=Yes or WrapAndShip=Yes, after generating output files Mif2Go copies distributable files to the directory designated by WrapPath (or in the case of JavaHelp or Oracle Help, a directory structure under the directory designated by WrapPath). If you want your final Help files to go somewhere other than subdirectory <code>_wrap</code>, change the value of WrapPath in your configuration file. See §7.2.4 Compiling and distributing Help systems on page 204.</p>

7.2.4 Compiling and distributing Help systems

After generating Help output files, **Mif2Go** can do the following:

- Create a directory (or a directory structure) for assembling the output.
- Copy the necessary files to the new directory or directory structure.
- Run the appropriate Help compiler, if there is one.
- Create a shipping directory.
- Archive the files required for distribution (not usually necessary for compiled Help).
- Place the compiled and/or archived Help system in the shipping directory.

See §35 [Producing deliverable results](#) on page 955.

See also:

§8.2.13 [Compiling a WinHelp project](#) on page 250

§9.14 [Compiling and testing HTML Help](#) on page 333

§10.13 [Assembling OmniHelp files for viewing](#) on page 369

§11.3.7 [Creating a directory structure for JavaHelp / Oracle Help](#) on page 378

§12.8 [Packaging Eclipse Help files](#) on page 419

7.3 Producing contents and index for Help systems

By default, **Mif2Go** generates both contents and index for Help systems:

- Contents entries are based on topic headings in your FrameMaker document.

- Index entries are produced from FrameMaker index markers.

However, you can choose to exclude contents or index or both.

In this section:

§7.3.1 [Understanding how Mif2Go produces contents and index](#) on page 205

§7.3.2 [Including FrameMaker TOC and IX in Help systems](#) on page 205

§7.3.3 [Grouping contents entries](#) on page 206

§7.3.4 [Modifying contents or index production for HTML-based Help](#) on page 206

§7.3.5 [Modifying contents or index production for WinHelp](#) on page 208

7.3.1 Understanding how Mif2Go produces contents and index

How **Mif2Go** generates contents (and index, for HTML-based Help) depends on which of the following you are converting:

[FrameMaker book](#)

[Chapter of a previously converted book](#)

[Single-file document](#)

*FrameMaker
book*

When you convert a FrameMaker book, **Mif2Go** produces the following:

- for WinHelp:
 - *.btc, a separate contents data file for each chapter file
- for HTML-based Help:
 - *.bhc, a separate contents data file for each chapter file
 - *.bhh, a separate index data file for each chapter file
 - MyDoc.lst*, a file that contains a list of the chapter files in book order

After all chapters are processed, **Mif2Go** combines entries from the chapter data files to create the final contents (and for HTML-based Help, the final index).

For HTML-based Help, **Mif2Go** recreates list file *MyDoc.lst* each time you run a conversion from the book file. **Mif2Go** uses *MyDoc.lst* to merge data files and produce the final contents and index files. See also §7.3.4.2 [Maintaining the list file when the book file changes](#) on page 208.

*Chapter of a
previously
converted book*

If the book file is open in FrameMaker when you convert a single chapter, **Mif2Go** produces from the chapter a contents data file (and for HTML-based Help, an index data file); then merges these data files with any from other chapters to regenerate the final contents (and index).

*Single-file
document*

When you convert a document that consists only of a single FrameMaker file, **Mif2Go** produces final contents (and index) directly, omitting intermediate data files.

7.3.2 Including FrameMaker TOC and IX in Help systems

Mif2Go produces contents and index for Help systems, so you do not need the TOC and IX files generated by FrameMaker. These files are excluded by default when you specify a Help system output type; see §5.5 [Converting FrameMaker-generated files](#) on page 124.

To include FrameMaker-generated TOC and IX files in a Help system:

```
[Setup]
; UseFrameTOC = Yes (default, except for Help formats),
;   or No (default for Help formats)
UseFrameTOC=Yes
; UseFrameIX = Yes (default, except for Help formats),
;   or No (default for Help formats)
UseFrameIX=Yes
```

If you do include these files, they become ordinary Help topics. If you include them and then change your mind after you have generated the Help system at least once, you must delete the resulting topic file(s) from the project directory, or they will remain in the Help project next time you generate output using **Mif2Go**.

For HTML-based Help, if you include TOC or IX files (or other FrameMaker-generated files) as topics, you must provide titles for them; for example:

```
[Titles]
MyProjTOC=Contents
MyProjIX=Index
```

See §13.4.5 [Specifying page titles for HTML output files](#) on page 433.

7.3.3 Grouping contents entries

Suppose you want to group the chapters in your document so that each group appears in the contents with its own “book” icon, in effect creating an additional contents level above the chapter level, and thus providing a more compact table of contents when only the first level is displayed in a Help viewer.

In your FrameMaker book (or a clone of the book intended for Help output), add a new chapter before the first chapter of each group. Give the titles of these new chapters a new format; for example, *PartTitle*. You can include summary text under the title, or leave the title as the only content.

In your project configuration file, edit the entries under [HelpContentsLevels] so that *PartTitle* is at a higher level than the chapter titles. If not all chapters are to be included in such a group, change the title format of the chapters that are in groups; for example, from *ChapterTitle* to *SubChapterTitle*; and give *SubChapterTitle* a setting that places it just below *PartTitle*. So you might have:

```
[HelpContentsLevels]
PartTitle=1
SubChapterTitle=2
ChapterTitle=1
...
```

You may need to adjust subhead levels also, if you have chapters at different contents levels.

7.3.4 Modifying contents or index production for HTML-based Help

You can use a configuration setting to override the default production method for contents or index or both; however, usually there is little reason to change the default method. You can also omit production of contents or index.

In this section:

§7.3.4.1 [Choosing contents and index methods for HTML-based Help](#) on page 207

§7.3.4.2 [Maintaining the list file when the book file changes](#) on page 208

§7.3.4.3 [Merging contents and index files from the command line](#) on page 208

See also:

§7.4 [Configuring contents entries for Help systems](#) on page 209

§7.5 [Configuring index entries for Help systems](#) on page 211

§9.9.1 [Choosing how to generate HTML Help contents and index](#) on page 319

§10.6 [Choosing navigation features for OmniHelp](#) on page 356

§11.4.5 [Locating JavaHelp or Oracle Help contents and index files](#) on page 387

§12.4.1 [Choosing contents and index methods for Eclipse Help](#) on page 411

7.3.4.1 Choosing contents and index methods for HTML-based Help

To specify whether contents, index, or both should be generated for HTML-based Help:

```
[MSHtmlHelpOptions] or
[OmniHelpOptions] or
[JavaHelpOptions] or
[OracleHelpOptions]
; ListType (for filter to create) = Both (default), Contents, or Index
```

To specify how contents and/or index should be generated for HTML-based Help, depending on the value of ListType:

```
[MSHtmlHelpOptions] or
[OmniHelpOptions] or
[JavaHelpOptions] or
[OracleHelpOptions]
; RefFileType = Full, Body, or None
```

RefFileType values have the following effects:

Body	<p><i>Default for books. Mif2Go creates the following:</i></p> <ul style="list-style-type: none"> • <i>MyDoc.lst</i>, if you are converting a book • <i>Chapter.bhc</i> for each chapter (if ListType=Contents or Both) • <i>Chapter.bhk</i> for each chapter (if ListType=Index or Both). <p>If you are converting a book, or if you are converting a chapter and the book file is open in FrameMaker, Mif2Go merges files to create the following:</p> <ul style="list-style-type: none"> • a combined contents file from all .bhc files (if ListType=Contents or Both) • a combined index file from all .bhk files (if ListType=Index or Both).
Full	<i>Default for single-file documents. Mif2Go creates final contents and index files directly, depending on the value of ListType.</i>
None	No list, contents, or index files are produced.

You might set RefFileType=None if you are repeatedly re-running a conversion to tune something in text, and you do not want the (small) overhead of writing out the contents and index information every time.

Note: If you set RefFileType=Full when you are converting a FrameMaker book, you will get contents and index entries only for the last chapter file in the book.

<i>HTML Help</i>	For HTML Help, you can specify one additional value for RefFileType. See §9.9.1 Choosing how to generate HTML Help contents and index on page 319.
<i>OmniHelp</i>	For OmniHelp, whether contents and index are displayed (as opposed to generated) is determined by another setting; see §10.6 Choosing navigation features for OmniHelp on page 356
<i>Eclipse Help</i>	For Eclipse Help, the default value of RefFileType is Full, regardless of whether you are converting a FrameMaker book or a single-file document. See §12.4.1 Choosing contents and index methods for Eclipse Help on page 411.

7.3.4.2 Maintaining the list file when the book file changes

For HTML-based Help, if you add a FrameMaker file to a book, or remove a file from a book, you must update *MyDoc.lst* (see §7.3.1 [Understanding how Mif2Go produces contents and index](#) on page 205). You can do this either of the following ways:

- rerun **Mif2Go** on the entire book
- use a text editor to add/remove file names, and rerun **Mif2Go** on added files.

If you delete *MyDoc.lst*, then generate output from individual chapter files, **Mif2Go** does not update the final contents or index. Final contents and index are not updated until you either convert the whole book or recreate *MyDoc.lst* some other way.

7.3.4.3 Merging contents and index files from the command line

For HTML-based Help, you can merge contents and index data files from the command line, as part of a batch process such as a product build. For example, to merge contents and index for HTML Help:

```
dc1 -fMB MyDoc.lst
```

See §37 [Converting via DCL](#) on page 995.

The first line in file *MyDoc.lst* must begin with LIST, and each subsequent line must name a FrameMaker .mif file included in the project, in the order in which the files should appear in the contents. You must execute the **dc1** command in the same directory as the .lst file.

See §37.2.5 [Merging ancillary Help files with DCL](#) on page 997.

7.3.5 Modifying contents or index production for WinHelp

Contents To specify how (and whether) contents entries are generated for WinHelp:

```
[HelpContents]
; CntType = None, Full (single file), or Body (headings, topics only)
```

CntType values have the following effects:

- | | |
|------|--|
| Body | <i>Default for books.</i> Mif2Go creates a .bct file for each chapter. If you are converting a book, or if you are converting a chapter and the book file is open in FrameMaker, Mif2Go merges *.bct files to create a combined contents file. |
| Full | <i>Default for single-file documents.</i> Mif2Go creates the final contents file, <i>MyDoc.cnt</i> , directly. |
| None | No .lst, .bct, or .cnt files are produced. |

Index To specify whether index entries are generated for WinHelp from FrameMaker index markers:

```
[HelpOptions]
; Index = Help (make into K footnotes) or None (removed)
```

See also:

§8.12 [Configuring index entries for WinHelp](#) on page 287

§8.13 [Configuring contents for WinHelp](#) on page 288

7.4 Configuring contents entries for Help systems

In this section:

§7.4.1 [Understanding how contents levels are assigned](#) on page 209

§7.4.2 [Setting contents levels for WinHelp](#) on page 209

§7.4.3 [Including contents entries in HTML-based Help](#) on page 209

§7.4.4 [Setting contents levels for HTML-based Help](#) on page 210

7.4.1 Understanding how contents levels are assigned

When you set up a **Mif2Go** Help project from FrameMaker, **Mif2Go** automatically assigns contents levels to heading paragraphs, as a first approximation. *Be sure to check these assignments*, and correct any that are not what you want. Contents levels are assigned to paragraph formats in the following configuration-file sections:

<u>Help type</u>	<u>Section</u>	<u>Reference</u>
WinHelp	[HelpCntStyles]	7.4.2
HTML-based Help	[HelpContentsLevels]	7.4.3

Levels in the contents might not correspond exactly to heading levels in your FrameMaker document, but they should be self consistent.

Avoid skipped contents levels

Hardly any Help systems allow skipped levels in the TOC. This usually happens when your FrameMaker document includes a sub subheading directly under a major heading, before the first subheading. Avoid this pattern; at the very least, exclude such sub subheadings from the TOC. By default, **Mif2Go** inserts the missing level, which works for some Help systems, but not for Eclipse Help. Skipped contents levels result in a warning in the **Mif2Go** log file.

7.4.2 Setting contents levels for WinHelp

For WinHelp, you specify the contents levels for your headings here:

```
[HelpCntStyles]
; format = H (heading), T (topic), or B (both), + level (1..9)
Heading 1=B2
```

See §8.13.2 [Specifying heading formats and levels for contents](#) on page 289.

Each entry in [HelpCntStyles] must correspond to an entry with property Contents in [HelpStyles]; for example:

```
[HelpStyles]
Heading 1=Topic Contents
```

See §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268.

See also:

§8.13 [Configuring contents for WinHelp](#) on page 288

7.4.3 Including contents entries in HTML-based Help

Headings that start topics are automatically included as contents entries in HTML-based Help. A paragraph format is included in contents when you assign any of the following to the format:

```
[HTMLParaStyles]
ParaFmt = Split
```



```
[HTMLParaStyles]
ParaFmt = Contents

[HelpContentsLevels]
ParaFmt = n
```

Avoid mid-topic links from TOC

It is best to assign the `[HTMLParaStyles]Split` property to every heading that you want to appear in the contents; see §18.2.1 [Designating split points](#) on page 586. Contents links to mid-topic locations can be problematic in some Help systems. For example, you cannot include mid-topic links in the TOC for HTML Help projects that are to be merged. In Java Help 1, the viewer cannot find mid-topic links at all.

Splitting on every heading provides faster loading of help topics, because they are shorter.

See also:

- §7.4.1 [Understanding how contents levels are assigned](#) on page 209
- §7.4.4 [Setting contents levels for HTML-based Help](#) on page 210
- §9.9.5 [Configuring contents entries for HTML Help](#) on page 322
- §10.7 [Configuring contents and index for OmniHelp](#) on page 357
- §11.4.1 [Configuring contents entries for JavaHelp or Oracle Help](#) on page 385
- §19.5.3 [Including ObjectID anchors as link targets](#) on page 620

7.4.4 Setting contents levels for HTML-based Help

To specify contents levels, assign level numbers to heading formats that start topics, and also to formats to which you have assigned the `Contents` property; see §7.4.3 [Including contents entries in HTML-based Help](#) on page 209. Level 1 is the top level; each higher level number represents another level of indentation in the TOC.

For example:

```
[HelpContentsLevels]
; FM paragraph format name = TOC level
PrefTitle = 1
ChapTitle = 1
AppxTitle = 1
Head1 = 2
Head2 = 3
Head3 = 4
AppxHead1 = 2
AppxHead2 = 3
```

You can specify any paragraph format, and all text in that format will appear in the table of contents. You are not restricted to paragraph formats that are mapped to HTML `Hn` tags (see §21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646).

Skipping a level between headings in your FrameMaker document can cause a TOC problem, especially for Eclipse Help and in Java Help. For example, suppose in one place in your document successive sections are organized like this:

```
Head1
    Head3
    Head2
    Head2
    Head3
```

A *Head3* section, which is at level 4, immediately follows a *Head1* section, which is at level 2. To correct this anomaly for the TOC, you would place a marker of type **HTMConfig** in the *Head3* paragraph, with the following content:

```
[HelpContentsLevels]=3
```


See §33.2 [Overriding settings with markers or macros](#) on page 920.

See also:

§7.4.1 [Understanding how contents levels are assigned](#) on page 209

7.5 Configuring index entries for Help systems

In this section:

§7.5.1 [Understanding how Mif2Go creates Help index entries](#) on page 211

§7.5.2 [Preparing index entries for Microsoft Help Viewer](#) on page 211

§7.5.3 [Limiting length of index entries for HTML Help or WinHelp](#) on page 212

§7.5.4 [Omitting intermediate index-range entries](#) on page 212

§7.5.5 [Treating commas as potential index level separators](#) on page 213

§7.5.6 [Combining index levels for HTML-based Help](#) on page 213

§7.5.7 [Configuring See and See also entries for HTML-based Help](#) on page 214

§7.5.8 [Specifying index link destinations for HTML-based Help](#) on page 215

§7.5.9 [Customizing index sort order](#) on page 216

See also:

§8.12 [Configuring index entries for WinHelp](#) on page 287

§9.9.8 [Customizing contents and index for HTML Help](#) on page 324

§10.7.6 [Redirecting See and See also index entries](#) on page 359

§11.4.3 [Configuring index entries for JavaHelp or Oracle Help](#) on page 386

7.5.1 Understanding how Mif2Go creates Help index entries

Mif2Go processes FrameMaker index markers to create the index for a Help system. Because **Mif2Go** operates only on catalogued formats, any character formats used in index markers must be present in the character catalog of each FrameMaker file where the index markers appear.

For HTML-based Help, **Mif2Go** usually builds the index, so you can customize several aspects of index organization.

For WinHelp, Microsoft [Help Workshop](#) always builds the index, so there is little **Mif2Go** can do to customize it.

7.5.2 Preparing index entries for Microsoft Help Viewer

Microsoft Help Viewer 1.x requires extra meta elements for index terms, so converting FrameMaker index markers to HTML or XHTML is not enough.

To have **Mif2Go** prepare index entries for eventual use in Microsoft Help Viewer 1.x:

```
[Index]
; UseHVIndex = No (default) or Yes, prepare meta elements for use
; with Microsoft Help Viewer 1.x
UseHVIndex = Yes
```

You must also remap index markers as follows:

```
[Markers]
Index = HVIndex
```

Mif2Go will also carry out normal index processing for HTML-based output if you clone index markers as well as remapping them:

```
[Markers]
Index = Index HVIndex
```

See §29.3.1 [Remapping and cloning marker types](#) on page 836.

When UseHVIndex=Yes, **Mif2Go** processes FrameMaker **HVIndex** marker content as follows:

- Removes the sequences <...> and [...].
- Converts commas to %2C.
- Converts colons to commas (with added space if needed).
- Passes through UTF-8 (from FrameMaker version 8 and later versions).
- Splits at semicolons.
- Respects backslash escapes.
- Produces five XML entities where needed.

Numeric character references are not needed, because **Mif2Go** uses UTF-8 for all non-ANSI characters.

Index terms for Microsoft Help Viewer 1.x look like this:

```
<meta name="Microsoft.Help.Keywords" content="Marker, Plain index" />
<meta name="Microsoft.Help.Keywords" content="Unicode%2C mañana..." />
<meta name="Microsoft.Help.Keywords" content="Index marker, First" />
<meta name="Microsoft.Help.Keywords" content="Index marker, Second" />
```

7.5.3 Limiting length of index entries for HTML Help or WinHelp

Mif2Go enforces a limit on length of index entries for the following Help systems:

[HTML Help](#)

[WinHelp](#)

HTML Help For HTML Help, the default limit is 488 characters:

```
[MSHtmlHelpOptions]
; KeywordLimit = 488 (default), max length of Help index entries
KeywordLimit=488
```

FrameMaker has a smaller effective limit, so the default setting is not likely to cause problems.

WinHelp For WinHelp, the default length is more conservative, because too many characters in a keyword can negatively affect the Help compiler. WinHelp documentation says the limit is 255 characters. **Mif2Go** sets the default to 64, based on experience:

```
[HelpOptions]
; KeywordLimit = max characters total (all levels) in keywords
KeywordLimit=64
```

You can test the limit yourself, watching for compiler errors as you increase the setting.

7.5.4 Omitting intermediate index-range entries

By default, **Mif2Go** produces a separate index entry for every topic that falls within an index range; that is, **Mif2Go** creates an index entry for each topic that occurs in your FrameMaker document between matching <\$startrange> and <\$endrange> index markers. The following limitations apply:

- If you have more than one identical <\$startrange> entry, the corresponding <\$endrange> entry terminates all matching <\$startrange> entries.

- If matching `<$startrange>` and `<$endrange>` entries are in different FrameMaker files, **Mif2Go** creates topic entries only up to the end of the file in which the `<$startrange>` entry occurs.

If your FrameMaker index includes broad ranges that span many topics (such as an entire chapter), you might not want every single topic title to appear in the generated index.

To omit all but the first and last topic references in each index range:

```
[HTMLOptions] or [HelpOptions]
; IndexRanges = Yes (default, include ref to every topic within range)
; or No (use only refs to topics at start and end of range in IX)
IndexRanges=No
```

You cannot eliminate the last entry in the range.

7.5.5 Treating commas as potential index level separators

By default, **Mif2Go** treats commas in FrameMaker index entries as potential level separators for indexes, even though commas serve only a grammatical function in a FrameMaker-generated index. However, **Mif2Go** breaks an index entry at a comma (or at any other level separator) only when there is at least one more entry that is an exact match up to the comma.

You can direct **Mif2Go** not to treat commas in index entries as level separators. The method depends on which type of Help system you are generating:

[HTML-based Help](#)

[WinHelp](#)

*HTML-based
Help*

To prevent use of commas as index level separators in HTML-based help:

```
[Index]
; UseCommaAsSeparator = Yes (default) or No (never break at comma)
UseCommaAsSeparator=No
```

When `UseCommaAsSeparator=Yes`, **Mif2Go** breaks an index entry either at a comma or at an unescaped colon to add another level, but only if the text of two or more such index entries match up to the comma or colon.

When `UseCommaAsSeparator=No`, **Mif2Go** does not break any index entry at a comma. Index entries are broken only at unescaped colons, and only when two or more entries match up to a colon; but also see §7.5.6 [Combining index levels for HTML-based Help](#) on page 213 and §7.5.7.2 [Specifying level breaks for See and See also index entries](#) on page 215.

WinHelp

To prevent use of commas as index level separators in WinHelp:

```
[HelpOptions]
; IdxColon = No (default, allow colon and comma as level delimiters)
; or Yes (use only colon as delimiter, treat comma as regular text)
IdxColon=Yes
```

See §8.12.1 [Designating index level separators](#) on page 287.

7.5.6 Combining index levels for HTML-based Help

Suppose you have a two-level index entry, with only one instance of a second level for the first-level text. In printed books, the usual practice is to combine the first- and second-level text into one first-level entry. By default, **Mif2Go** follows this practice for HTML-based Help indexes. If you expect to merge indexes from two or more Help projects, you might not want the levels combined.

To keep **Mif2Go** from combining levels for such index entries:

```
[Index]
; CombineIndexLevels = Yes (default) or No (always break at colon)
CombineIndexLevels=No
```

When `CombineIndexLevels=Yes`, if there is only one item at the last level of a series of multi-level index entries, **Mif2Go** removes the colon before the last item and replaces it with one of the following, depending on what character immediately precedes (or follows) the level-break colon:

- *nothing*, if a space precedes or follows the colon
- a *space*, if punctuation (such as an escaped colon or a comma) precedes the colon
- a *comma* followed by a *space*, if an alphanumeric character precedes the colon.

For example, suppose an index marker contains:

```
tomatoes:Cherokee Purple
```

and there are no other first-level index entries for “tomatoes”. This entry would become:

```
tomatoes, Cherokee Purple
```

in the Help index.

When `CombineIndexLevels=No`, **Mif2Go** breaks index entries at all unescaped colons.

7.5.7 Configuring See and See also entries for HTML-based Help

If you tell **Mif2Go** the words used to introduce redirect references in your FrameMaker index entries, **Mif2Go** can use this information to sort entries, to determine index level breaks, and (for some Help systems) to create live links to the referenced index entries.

In this section:

[§7.5.7.1 Identifying See and See also index references](#) on page 214

[§7.5.7.2 Specifying level breaks for See and See also index entries](#) on page 215

[§7.5.7.3 Choosing where to sort See also index references](#) on page 215

7.5.7.1 Identifying See and See also index references

To make sure **Mif2Go** recognizes *See* and *See also* entries in your FrameMaker index, you can specify the words used as starting terms in those entries. **Mif2Go** uses this information for sorting (see [§7.5.7.3 Choosing where to sort See also index references](#) on page 215) and for merging (see [§7.11 Setting up a dynamic modular Help system](#) on page 241).

To specify the words used for *See* and *See also* in your FrameMaker index:

```
[Index]
; SeeTerm = word(s) used as the start of a See index entry, default
; "See" without the quotes, case is significant
SeeTerm=See
; SeeAlsoTerm = word(s) used as the start of a See also entry, default
; "See also" without the quotes, case is significant
SeeAlsoTerm = See also
```

These settings allow you to designate different terms, perhaps in another language, or to specify a different case. For example, if you always capitalize both words in your *See also* index entries:

```
[Index]
SeeAlsoTerm = See Also
```

Mif2Go recognizes *See* and *See also* entries that include multiple references, provided the references are separated by semicolons. For example:

pome fruits, *see* apples; pears

includes two references, one to the index entry for “apples” and one to the entry for “pears”.

*Links are active in
OmniHelp and
HTML Help*

For OmniHelp, **Mif2Go** redirects both *See* and *See also* links to their targets in the index itself; see §10.7.6 [Redirecting See and See also index entries](#) on page 359. Microsoft HTML Help Workshop also redirects these links. However, there may be no visual clue that the links are there; you might have to double-click an index entry to activate the link.

Note: *See/See also* references that are not an exact match to a level 1 index term are omitted.

7.5.7.2 Specifying level breaks for *See* and *See also* index entries

By default, **Mif2Go** forces an index level break for a *See* or *See also* index entry. For example, this index entry in FrameMaker:

pome fruits, *see* apples; pears

would become:

pome fruits
 see apples; pears

To prevent arbitrary index level breaks for *See* and *See also* entries

```
[Index]
; LevelBreakForSee = Yes (default, always force a level break before
; See and See also entries), or No (break only for explicit colon)
LevelBreakForSee=No
```

When `LevelBreakForSee=No`, a level break occurs for a *See* or *See also* index entry only if an unescaped colon precedes the *See* or *See also* clause. See §7.5.5 [Treating commas as potential index level separators](#) on page 213.

7.5.7.3 Choosing where to sort *See also* index references

By default, **Mif2Go** places *See also* references last at any given index level. However, you can change the sort order so that *See also* references come *first* at any given index level.

To make *See also* entries sort first, ahead of other entries at the same index sublevel:

```
[Index]
; SortSeeAlsoFirst = No (default, put See also entries after any other
; index subentries), or Yes (put them first after the parent entry)
SortSeeAlsoFirst = Yes
```

Only OmniHelp and HTML Help actually honor this setting; JavaHelp and Oracle Help ignore it.

7.5.8 Specifying index link destinations for HTML-based Help

`KeywordRefs` applies only when **Mif2Go** generates index entries for Help systems, based on markers in your FrameMaker document. This setting does not apply when **Mif2Go** converts an existing FrameMaker index.

To specify where an index-entry link should point:

```
[Index]
; KeywordRefs = Keyword (default), File, or Para (at start)
KeywordRefs = Keyword
```

Selecting an item in the resulting index takes you to one of the locations listed in [Table 7-1](#), depending on the setting you specified for `KeywordRefs`.

Note: `KeywordRefs` applies only when **Mif2Go** generates index entries for Help systems, based on markers in your FrameMaker document. This setting does not apply when **Mif2Go** converts an existing FrameMaker index.

Table 7-1 Index link options for KeywordRefs in HTML-based Help

Option	Destination format	Location with respect to index marker
Keyword	<code>topicfile.htm#objectID</code>	Exact location of the marker; works well when index markers are placed just before the words they reference.
Para	<code>topicfile.htm#objectID</code>	Start of the paragraph containing the marker; use when other index markers occur in the same paragraph.
File	<code>topicfile.htm</code>	Start of the file containing the marker; use if all index markers are at the end of their topics; also for WebHelp and for merged HTML Help CHM files.

When you specify `KeywordRefs=Keyword` or `KeywordRefs=Para`, **Mif2Go** generates index link destinations of the following form:

```
topicfile.htm#objectID
```

When `KeywordRefs=Para`, also make sure you use the following (default) setting:

```
[HtmlOptions]
ObjectIDs=All
```

Otherwise, some or all mid-topic targets might be missing from the generated HTML. See §13.8.2.2 [Including paragraph references](#) on page 445 and §19.5.3 [Including ObjectID anchors as link targets](#) on page 620.

*RoboHelp lacks
mid-topic index
links*

RoboHelp does not recognize the mid-topic hash (fragment) identifiers that specify mid-topic index destinations. Therefore, you must use the following setting if you plan to use RoboHelp to generate WebHelp:

```
[Index]
KeywordRefs=File
```

As a result, index links always put you at the beginning of the referenced topic in WebHelp.

*Merged CHM files
cannot use index
anchors*

When an index entry in HTML Help points to more than one topic, the viewer displays a *Topics Found* dialog box that lists the topics by name. However, in merged CHM files, if the index entries for a slave file point to anchored locations (`topicfile.htm#anchor`), the *Topics Found* dialog box displays the index entry instead of the destination. To avoid this problem, use the following setting:

```
[Index]
KeywordRefs=File
```

7.5.9 Customizing index sort order

For stand-alone (unmerged) HTML Help and for OmniHelp, **Mif2Go** can control the order of index entries and subentries. Merged HTML Help requires a binary index, which ignores **Mif2Go** settings. JavaHelp and Oracle Help tend to ignore any sort order **Mif2Go** produces.

In this section:

§7.5.9.1 [Listing characters to ignore in index sort order](#) on page 217

§7.5.9.2 [Choosing case sensitivity of indexed terms](#) on page 217

§7.5.9.3 [Specifying index sort type and locale](#) on page 218

§7.5.9.4 [Choosing whether to use FrameMaker index sort strings](#) on page 218

§7.5.9.5 Defining Japanese index sort order on page 218

See also:

§7.5.7.3 Choosing where to sort See also index references on page 215

7.5.9.1 Listing characters to ignore in index sort order

To specify which characters **Mif2Go** should ignore when ordering index entries for HTML Help or OmniHelp, use one or both of the following settings:

```
[Index]
; IgnoreCharsIX = characters to exclude when sorting index entries
; (en dash, em dash, and nonbreaking hyphen are all converted to
; hyphens first)
IgnoreCharsIX=-[ ](<>_
; IgnoreLeadingCharsIX = characters to exclude if at the beginning of
; the entry when sorting index entries; multiples like $$ or .. are
; all excluded
IgnoreLeadingCharsIX=.$
```

By default, when sorting index entries **Mif2Go** ignores the following characters:

- anywhere in an entry:
 - hyphen, nonbreaking hyphen, en dash, em dash
 - [] left and right square brackets
 - () left and right parentheses
 - < > left and right angle brackets
 - _ **underscore.**
- as the leading character(s) in an indexed term:
 - . period(s)
 - \$ dollar sign(s).

If you do not include *any* settings for IgnoreCharsIX or IgnoreLeadingCharsIX, **Mif2Go** uses these defaults. Characters specified for IgnoreCharsIX affect the sorting of sublevels; those specified for IgnoreLeadingCharsIX do not.

Suppose you provide no setting at all for IgnoreCharsIX, and just specify this setting:

```
[Index]
IgnoreLeadingCharsIX=?
```

In this case all of the following characters would be ignored for index sorting:

- any number of ? at the beginning of any indexed term
- any number of -, [], (), <, >, or _ anywhere in any entry.

To have *only* leading question marks ignored, you would specify:

```
[Index]
IgnoreCharsIX=
IgnoreLeadingCharsIX=?
```

To exclude *all* characters from the “ignore” sets, so all index entries that start with punctuation appear at the beginning of the Help index:

```
[Index]
IgnoreCharsIX=
IgnoreLeadingCharsIX=
```

7.5.9.2 Choosing case sensitivity of indexed terms

If your FrameMaker document is heavily indexed on case-sensitive terms, you might want to make sure the Help index keeps terms separate if they differ only in case:


```
[Index]
; CaseSensitiveIndexCompare=No (default)
; or Yes (treat words that differ only in the case of their first
; letter as different)
CaseSensitiveIndexCompare=Yes
```

The default is to ignore case sensitivity, which can cause terms that differ only in the case of the first letter to be grouped in the Help index as though they are the same term.

7.5.9.3 Specifying index sort type and locale

To ensure that accented or non-Western characters sort the way you want them to in the index, you might have to specify a different sort type, or a non-English locale:

```
[HtmlOptions]
; IndexSortType = Numeric (default, code-point order),
; Lexical (using MS strcoll functions), or
; Alpha (sort accented letters as though they are unaccented).
IndexSortType=Numeric
; IndexSortLocale = language to use for sorting index.
; When IndexSortType is Lexical, default is current
; OS country setting. Uses MS language names.
;IndexSortLocale=English
```

For example, to make accented letters sort as though unaccented, specify the following:

```
[HtmlOptions]
IndexSortType = Alpha
```

Alpha works only with the Windows Western character set and the Unicode character set. Specify **Lexical** for Central European or Cyrillic locales; Alpha does not handle those. However, **Mif2Go** does support multibyte sorting when you specify the index locale.

If you use **Mif2Go** to produce HTML Help in an Asian or Cyrillic language, also specify the Help-file language; see §9.13 [Generating HTML Help in non-Western languages](#) on page 331.

7.5.9.4 Choosing whether to use FrameMaker index sort strings

The sort strings you can include in brackets in FrameMaker index markers (for example, [peaches:aaa]) rarely work well for Help systems. By default, **Mif2Go** ignores any sort strings present in index markers, and generates sort strings anew. Although you can direct **Mif2Go** to use FrameMaker sort strings, we do not recommend it.

To make **Mif2Go** use FrameMaker index sort strings:

```
[Index]
; UseSortString = No (default)
; or Yes (use Frame sort string if present)
UseSortString = Yes
```

When **UseSortString=Yes**, **Mif2Go** honors sort strings in FrameMaker index markers. When **UseSortString=No**, **Mif2Go** ignores FrameMaker sort strings, and instead generates new sort strings for Help index entries.

Sort strings are not allowed at all in Microsoft Help Viewer 1.x. This means it is impossible to get a proper Asian sort, especially for Japanese.

7.5.9.5 Defining Japanese index sort order

Japanese is not sorted by the Unicode character order of the displayed glyphs, which are usually katakana. Instead, it is sorted according to the pronunciation of the words, as given in a different script, kanji. **Mif2Go** has no idea what the kanji is, so you have to provide it

by the usual FrameMaker sort order method, in square brackets for every single index entry. Naturally this requires a Japanese native speaker. Once you have all the kanji, and have the Japanese version of the SortOrderIX entry on the FrameMaker index reference page, you should be able to produce a sorted index in Japanese.

You might be able to convert the katakana to kanji with software assistance; see:

<http://kakasi.namazu.org/>

If you are adept at Perl programming, you might be able to adapt the following Perl module for this purpose:

<http://search.cpan.org/~dankogai/Text-Kakasi-2.04/Kakasi.pm>

7.6 Providing related-topic links for Help systems

You can include links to related topics several ways. According to usability studies, the best way is also the simplest: create lists of cross references (perhaps under a *Related Topics* heading) in your FrameMaker document files. **Mif2Go** converts these cross references to links.

Mif2Go also supports dedicated related-topic links:

- associative links (ALinks) for all Help systems except WinHelp 3 and JavaHelp
- keyword links (KLinks) for HTML Help and OmniHelp.

Dedicated related-topic links are typically displayed in a menu or pop-up window, or in a navigation pane. With limited screen estate, they offer the advantage of not cluttering the topic pane with *See* and *See also* entries.

In this section:

§7.6.1 [Understanding related-topic links](#) on page 219

§7.6.2 [Understanding how ALinks work](#) on page 220

§7.6.3 [Understanding how KLinks work](#) on page 221

§7.6.4 [Adding related-topic link keywords in FrameMaker](#) on page 221

§7.6.5 [Adding ALink and KLink jumps in FrameMaker](#) on page 222

§7.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 224

§7.6.7 [Specifying ALink and KLink list-link destinations](#) on page 224

7.6.1 Understanding related-topic links

Related-topic links you can produce with **Mif2Go** come in two flavors:

- associative link (ALink)
- keyword link (KLink).

Each consists of a jump from one topic to a list of links to other topics. The listed links are members of a set of links that share a common identifier, or *link keyword*. KLink keywords are actually index entries, while ALink keywords are subject terms embedded in the member topics. ALink keywords are not ordinarily visible to users, except in OmniHelp (see §10.8 [Providing related-topic links in OmniHelp](#) on page 359).

Link keywords

You embed ALink keywords in target topics, using FrameMaker markers or dedicated FrameMaker formats; KLink keywords are already present, in the form of index markers. ALink and KLink keywords are case sensitive. Each ALink keyword must consist of a single alphanumeric term. Punctuation is not allowed; however, spaces are allowed in ALink keywords in some Help systems.

<i>Related-topic jumps</i>	You insert ALink and KLink jumps with FrameMaker hypertext Go to URL markers (see §34.1.2 Using markers to add links and instructions on page 935); or, you can roll your own using Mif2Go macros (for HTML-based Help) or WinHelp macros (for WinHelp 4).
<i>Bullet-proof links</i>	Why use ALinks and KLinks if your document already includes cross references, <i>See also</i> lists, and other hypertext links? Unlike other links, an ALink or KLink jump can go (via the list of links) to multiple target topics, yet does not <i>require</i> the presence of any topic. Therefore, you can do the following without disturbing related-topic links: <p style="margin-left: 40px;"><i>Add a topic:</i> Existing ALink and KLink jumps automatically pick up any relevant link keywords in the new topic.</p> <p style="margin-left: 40px;"><i>Remove a topic:</i> If no link keywords exist in the remaining topics for a given ALink or KLink jump, instead of triggering an error message, the jump does nothing.</p>
<i>Run-time activation</i>	ALinks and KLinks are especially useful if you expect to merge Help projects (see §7.11 Setting up a dynamic modular Help system on page 241), for the following reasons: <ul style="list-style-type: none"> • If other Help projects are merged with the main project at run time, and topics in the merged projects contain KLink or ALink keywords that appear in the main project, links to those topics are included in the relevant ALink and KLink lists in the main project. • If a section (or a whole subproject) is not found at run time because it was not installed, any ALink or KLink references to topics in that section quietly disappear from the main project, whereas regular links would yield error messages.
<i>KLinks access merged topics</i>	If you merge your Help project with another Help project built by someone else, possibly using other tools, KLinks can provide the only way to add links to topics in the other project, assuming the other project has a thorough index.

7.6.2 Understanding how ALinks work

Associative links (ALinks) connect a given topic to one or more other topics that share the same ALink keyword. When you are viewing a topic that contains an ALink jump, you can click the ALink jump hotspot (or related-topics button) to see a list of links to associated topics. The list of links is displayed either in a navigation pane (as in OmniHelp), or in a pop-up menu or dialog (as in WinHelp 4, HTML Help, and Oracle Help for Java).

The following **Mif2Go**-generated Help systems support ALinks:

WinHelp 4
HTML Help
OmniHelp
Oracle Help for Java

<i>ALink keyword</i>	You insert an ALink keyword in a topic, to accomplish the following: <ul style="list-style-type: none"> • assign membership of that topic in an ALink set • provide a destination for corresponding links from an ALink list, which is accessed from an ALink jump. <p>The ALink set is identified by the ALink keyword. In effect, you label the topic with an ALink keyword.</p>
<i>ALink jump</i>	In some other topic, you insert an ALink jump that specifies the same ALink keyword; when a user clicks that ALink jump, the corresponding ALink list of links to all topics in the set is displayed.

*Bi-directional
ALinks*

OmniHelp supports a variation: all topics that share the same ALink keyword belong to the same “pool” of topics. When any topic that is a member of a pool is displayed, links to all other members of that pool are automatically listed in the navigation pane when you click **Related**. See §7.6.4.1 [Adding related-topic link keywords via markers](#) on page 221.

For a similar approach in HTML Help and Oracle Help for Java, see §7.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 224.

7.6.3 Understanding how KLinks work

Keyword links (KLinks) are based on index entries. When you are viewing a topic that contains a KLink jump, you can click the jump hotspot (or related-topics button) to display a list of links to all topics that are indexed on the keyword(s) specified in the jump.

The following **Mif2Go**-generated Help systems nominally support KLinks, though only for index entries that meet assorted restrictions:

HTML Help
OmniHelp
WinHelp 4

You insert in a topic a KLink jump that specifies the content of one or more entries in the index. When a user selects the KLink jump, all index entries with the same content, and with the same links as in the index, are displayed in a list.

*Use KLinks only
as a last resort*

KLinks are high-maintenance items for documents where index entries are subject to change when the document is revised. An index term in a KLink jump must match exactly a term in the index itself; if the term is changed in the index, you must make the identical change in any KLink jump that references that index term, or the jump will not generate a link to the corresponding topic; and in some systems, it might yield an error message instead. Help-system implementation of KLinks is uneven. KLinks have proved to be problematic in all **Mif2Go**-generated Help systems where they are nominally supported.

7.6.4 Adding related-topic link keywords in FrameMaker

In this section:

§7.6.4.1 [Adding related-topic link keywords via markers](#) on page 221

§7.6.4.2 [Adding related-topic keywords via format properties](#) on page 222

See also:

§7.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 224

7.6.4.1 Adding related-topic link keywords via markers

You can use FrameMaker **ALink** markers to insert ALink keywords. The content of the marker is an ALink keyword that identifies the ALink set to which the topic belongs. An ALink keyword is case sensitive, and must consist of a single alphanumeric term, without punctuation. However, in OmniHelp (only), spaces are allowed in ALink keywords, and you can include as many keywords as you want in a single marker, separated by semicolons.

If your FrameMaker document already has an index, you do not need to insert keywords for KLink jumps; the keywords are already present in your FrameMaker index markers.

You can provide markers to use for ALink keywords in any of the following ways:

- Add custom marker type **ALink** to your FrameMaker document; see §29.2 [Adding custom marker types](#) on page 832.

- Clone an existing marker type; see §5.11 [Repurposing FrameMaker markers](#) on page 139
- For HTML-based Help only, redefine the behavior of an existing marker type; see §29.4 [Defining and redefining marker behavior](#) on page 838.

For example, to use **Subject** markers for ALinks, you could specify the following:

```
[Markers]
Subject = ALink
```

Then, any **Subject** markers in your document whose content conforms to the requirements for ALink keywords can serve as ALink keyword markers. To make those **Subject** markers serve as KLink keyword markers also, you could specify the following:

```
[Markers]
Subject = ALink Index
```

Then, the content of any **Subject** markers in your document would also show up in the FrameMaker index (if any), and would be included in KLink lists.

Note: For WinHelp, you need additional settings for ALink keyword markers; see §8.11.2 [Adding ALinks and KLinks with markers](#) on page 285.

7.6.4.2 Adding related-topic keywords via format properties

You can assign a property to a FrameMaker format to make the text of every instance of that format act as a related-topic keyword. Create and catalog a special format to use for this purpose.

WinHelp For WinHelp, you can use either a character format or a paragraph format for related-topic keywords. For example, if an ALink keyword (“A” footnote) appears as a word in topic text, you can apply a special character format to the word, and in the configuration file assign property AKey to the format:

```
[HelpStyles]
ALinkCharFmt = AKey
```

Or, you can insert an ALink keyword in a paragraph by itself, apply a special paragraph format (or special character format, provided you do not use the *same* format to mark words in topic text), and also assign property Delete to the format:

```
[HelpStyles]
ALinkParaFmt = AKey Delete
```

See §8.11 [Creating related-topic links in WinHelp](#) on page 285.

HTML-based Help For HTML-based Help you must use a paragraph format (as opposed to a character format) for related-topic keywords. For example, you can put an ALink keyword in a paragraph by itself, apply a special paragraph format, and in the configuration file assign property ALink to the format (and property Delete, if you do not want the paragraph to appear in topic text):

```
[HTMLParaStyles]
; ALink, effective for MS HTML, OH, and Oracle Help, uses the contents
; of the para for the value of the ALink Name parameter of an ALink
; object.
ALinkParaFmt = ALink Delete
```

7.6.5 Adding ALink and KLink jumps in FrameMaker

Use hypertext **Go to URL** markers to insert ALink and KLink jumps in your document. See §34.1.2 [Using markers to add links and instructions](#) on page 935.

In this section:

§7.6.5.1 [Configuring ALink jumps](#) on page 223

§7.6.5.2 [Configuring KLink jumps](#) on page 223

See also:

§5.10 [Creating hotspots for hypertext links](#) on page 138

§7.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 224

7.6.5.1 Configuring ALink jumps

An [ALink](#) jump specifies one or more ALink keywords; clicking an ALink jump hotspot displays a list of links to any other topics that contain any of the same ALink keywords. Some restrictions:

- WinHelp and Oracle Help for Java restrict each ALink jump to a single ALink keyword.
- HTML Help and Oracle Help for Java restrict each ALink keyword to a single word (no spaces).

To add an ALink jump in FrameMaker, insert a hypertext **Go to URL** marker with content like the following:

```
message URL alink:keyword
```

No spaces are allowed after the colon that follows `alink`. URL must be capitalized.

If you specify multiple ALink keywords (which you can for OmniHelp or HTML Help), separate the identifiers with semicolons (no spaces!). For example:

```
message URL alink:curry;chicken;turmeric
```

7.6.5.2 Configuring KLink jumps

A [KLink](#) jump can specify one or more index terms; this should give you, in effect, one or more links to any other topics for which there are index entries that consist of those terms. In FrameMaker, that would be any other topic that contains an **Index** marker (or another type of marker cloned to **Index**) that has the same content as a term specified in the KLink jump.

JavaHelp and Oracle Help for Java do not support KLink jumps. Although HTML Help, OmniHelp, and WinHelp 4 nominally support KLink jumps, the jumps actually work only in restricted circumstances. KLink jumps are problematic at best, and should be tested individually.

To add a KLink jump in FrameMaker, insert a hypertext **Go to URL** marker with content like the following:

```
message URL klink:index term1;index term2;...
```

URL must be capitalized. Separate index terms from each other with semicolons (no spaces). Index terms can contain spaces; however, no spaces are allowed after the colon that follows `klink`.

*Commas or
colons for level
separators*

Mif2Go treats commas and colons in FrameMaker index markers as exactly equivalent, and changes all colons to commas. If there is no space after a colon or comma, **Mif2Go** adds a space. After changing colons to commas and sorting index entries, **Mif2Go** treats commas as level separators if two successive entries match through a comma, or if the shorter entry ends at a comma in the longer entry.

*Exact match
required*

The text of each index term in a KLink jump must match exactly, including case, the text of the corresponding entry in the index, with the following restrictions:

- **Escape double quotes.** If an index entry contains double quotes, you must escape each double quote with a backslash in the KLink jump
- **Eschew semicolons as punctuation.** Because semicolons are *always* index-term separators, a KLink jump cannot specify an index term that contains a semicolon; the semicolon cannot be escaped with a backslash.

7.6.6 Creating target-and-jump ALinks for HTML-based Help

You can insert ALink information in FrameMaker that serves as both an ALink-list target and an ALink jump, so that all ALink instances with the same keyword belong to a “pool” of ALinks; clicking any one of them displays a list of links to all other topics that have the same keyword.

<i>OmniHelp</i>	For OmniHelp, just inserting ALink keyword markers has this effect. Whenever the OmniHelp navigation control is set to Related , if the currently displayed topic contains an ALink keyword marker, the navigation pane displays a list of links to all other topics that contain ALink markers with the same keyword.
<i>HTML Help, Oracle Help for Java</i>	For HTML Help or Oracle Help for Java, you use a paragraph format to specify ALink keywords, and supply macro code to surround the paragraph for the ALink jump. When you assemble ALink jumps using macros, you are not making use of any Mif2Go code to interpret the <code>alink</code> protocol; whatever you build is passed through to the Help system, unaltered.
<i>HTML Help</i>	For HTML Help, the ALink jump code can produce a button; see §9.7.4 Rolling your own macros for ALink jumps in HTML Help on page 312.
<i>Oracle Help for Java</i>	For Oracle Help for Java, the ALink jump code creates a hotspot; see §11.10 Creating ALinks for Oracle Help on page 399.

7.6.7 Specifying ALink and KLink list-link destinations

For WinHelp 4, HTML Help, and OmniHelp, links from related-topic lists always go to the beginning of the topic.

For Oracle Help for Java, you can determine whether ALinks go to the beginning of the referenced topic file, or to the beginning of the paragraph that contains the ALink keyword; see §11.10 [Creating ALinks for Oracle Help](#) on page 399.

7.7 Jumping to secondary windows in Help systems

To cause a jump to go to a secondary window, assign the name of the target window to the character or paragraph format you use for the jump hotspot. Any jumps from text in the specified format go to the window you assigned rather than to the current window.

In this section:

§7.7.1 [Assigning secondary windows for WinHelp](#) on page 224

§7.7.2 [Assigning secondary windows for HTML-based Help](#) on page 225

7.7.1 Assigning secondary windows for WinHelp

For WinHelp, assign the name of a secondary window to a hotspot paragraph or character format in [HelpWindowStyles], and assign the Window property to the same format in [HelpStyles]; for example:

```
[HelpStyles]
JumpToExtra = JumpHot Green Window
```



```
[HelpWindowStyles]
JumpToExtra = extra
```

See §8.9.7 [Specifying jumps to secondary windows in WinHelp](#) on page 277.

7.7.2 Assigning secondary windows for HTML-based Help

For HTML-based Help, assign the name of a secondary window to a hotspot paragraph or character format in [SecWindows]; for example:

```
[SecWindows]
; doc format = name of secondary window to use for jumps from
; within the span marked by this style (same as WinHelp usage).
ProcWin = proc
```

For Oracle Help, JavaHelp, and OmniHelp, reserved window name `Popup` specifies a pop-up window; see §7.8 [Creating pop-up topics for Help systems](#) on page 225. For OmniHelp, you can include optional window parameters in the assignment.

See also:

§9.8 [Using secondary windows in HTML Help](#) on page 317

§10.9 [Jumping to secondary windows in OmniHelp](#) on page 360

§11.8.3 [Jumping to secondary windows in JavaHelp or Oracle Help](#) on page 399

§19.4 [Creating jumps to particular windows for HTML](#) on page 616

7.8 Creating pop-up topics for Help systems

You can use **Mif2Go** to create pop-up topics in any of the Help formats.

In this section:

§7.8.1 [Understanding pop-up hotspots, links, and topics](#) on page 225

§7.8.2 [Defining a pop-up hotspot](#) on page 226

§7.8.3 [Displaying a topic in a pop-up window](#) on page 226

7.8.1 Understanding pop-up hotspots, links, and topics

In FrameMaker, you delimit a pop-up *hotspot* by applying a dedicated character or paragraph format to text from which the topic is to be accessed. You provide a link to the pop-up topic with either of the following:

- a cross reference (except for HTML Help) from the hotspot
- a hypertext link inserted in the hotspot.

Do not place any other markers within the hotspot area.

Properties you assign to the hotspot format cause a new window to pop up, displaying the referenced topic, when you click the hotspot.

Except in HTML Help, you define pop-up topics like any other topics; only the way they are displayed makes them different from “normal” topics.

Note: If you are using JavaHelp 2 to view this information, the only active part of the hotspot is an icon that immediately precedes the hotspot text. See §11.8.1.4 [Specifying window-access object properties](#) on page 395.

7.8.2 Defining a pop-up hotspot

To define a hotspot, in FrameMaker apply a dedicated character or paragraph format to either of the following:

- the text of a cross reference
- text that contains a hypertext link.

If you want content (or an autonumber, or a page reference) from the pop-up material to appear as a hotspot, use a cross reference; if not, use a hypertext link.

Cross reference If you use a cross reference for a hotspot, make sure the hotspot character format is applied to the reference string (<\$paratext>) in the cross-reference definition. If another character format intervenes (for example, if your cross-reference format is defined as <HotspotFmt><Bold><\$paratext>), the intended hotspot format is turned off before it can be used.

Entire paragraph (by accident) If you do not apply a character format to hotspot text for a hypertext link, *the entire paragraph becomes a hotspot*.

Entire paragraph (on purpose) If you use a paragraph format (such as the text of a generated-file reference) for a hotspot, and some of the text in the paragraph has a character format applied, you must assign property `ParaLink` to the paragraph format.

See §5.10 [Creating hotspots for hypertext links](#) on page 138.

7.8.3 Displaying a topic in a pop-up window

WinHelp For WinHelp, to make a topic pop up, assign property `PopOver` to the hotspot format:

```
[HelpStyles]
PopCharFmt = PopOver
```

See §8.9 [Creating jumps and pop-ups for WinHelp](#) on page 272.

HTML-based Help If your HTML-based Help system has a **Mif2Go**-generated browse sequence, to avoid including pop-up topics in the browse sequence you must declare these topics to be extracts instead of splits; see §18.3 [Extracting files](#) on page 591.

HTML Help For HTML Help, all you get is plain-text pop-ups, unless you use a third-party tool. To create a pop-up link in HTML Help, put the entire pop-up content (plain text only) in a FrameMaker hypertext **alert** marker embedded in the hotspot.

See §9.5 [Creating pop-ups for HTML Help](#) on page 305.

OmniHelp, JavaHelp, Oracle Help for java For OmniHelp, JavaHelp, and Oracle Help for Java, you can use any HTML in pop-up topics, including graphics and jumps. To make a topic pop up, assign reserved window name `popup` to the hotspot format:

```
[SecWindows]
PopCharFmt = popup
```

See:

§10.9 [Jumping to secondary windows in OmniHelp](#) on page 360

§11.8 [Defining windows for JavaHelp or Oracle Help](#) on page 393.

7.9 Including expandable sections in Help topics

For OmniHelp and HTML Help (and for HTML and XHTML), you can use a combination of JavaScript and **Mif2Go** macros to create one or more expandable drop-down sections in a topic.

In this section:

- §7.9.1 [Understanding Mif2Go expandable drop-down sections](#) on page 227
- §7.9.2 [Setting up expandable sections for your document](#) on page 227
- §7.9.3 [Delimiting expandable drop-down sections](#) on page 228
- §7.9.4 [Configuring drop-down links](#) on page 230
- §7.9.5 [Configuring drop-down blocks](#) on page 233
- §7.9.6 [Providing CSS for drop-down links and blocks](#) on page 233
- §7.9.7 [Deploying JavaScript code for drop-down sections](#) on page 234
- §7.9.8 [Emulating Web Works Publisher drop-down hotspots](#) on page 237

7.9.1 Understanding Mif2Go expandable drop-down sections

An expandable drop-down section allows a user to click a link to optionally display additional material; then click the link again (or click the displayed material) to collapse the section and hide the material.

For **Mif2Go** output, the link can be based on any of the following:

- a special paragraph format dedicated to drop-down links
- an existing paragraph format in your document, such as a figure title or table title
- a graphic icon, with or without accompanying text
- a button, instead of text
- a fixed text string.

A drop-down section has four main parts: link start, link end, block start, and block end. Each new drop-down link gets a new value for predefined macro variable `<$$_DropID>`, which is used in all following blocks until the next link, or until the end of the HTML file. This means that a single link can optionally control multiple blocks; the blocks do not have to be contiguous.

Each link/block set is independent of other sets. Opening one block does not close other blocks that might have been opened from other links.

Mif2Go provides built-in macros to use for drop-down sections, and settings to enable and deploy the macros. To use the built-in **Mif2Go** macros *as is* for drop-down sections, you do not have to include their definitions in your configuration file. Include a drop-down macro definition only to edit or replace the macro.

In its simplest form, a **Mif2Go** drop-down section needs only one `[HTMLParaStyles]` format property assigned in the configuration file, and rarely more than two; but has enough configurable options to do almost anything you might want.

7.9.2 Setting up expandable sections for your document

To enable expandable drop-down sections in the HTML output from your FrameMaker document:

```
[DropDowns]
; UseDropDowns = No (default) or Yes (enable use of dropdowns)
UseDropDowns = Yes
```

Simple drop-down sections

To provide simple drop-down sections:

- Use a dedicated paragraph format for material you want to allow users to expand.
- Assign property `DropDown` to the format.
- Let **Mif2Go** supply the drop-down links.

For example, suppose you apply paragraph format *ExpandThis* to each paragraph to be expanded. You would include the following setting in your project configuration file:

```
[HTMLParaStyles]
ExpandThis = DropDown
```

With this setting, **Mif2Go** would insert a drop-down link in HTML output just before each *ExpandThis* paragraph, and hide the paragraph when a user first displays the topic. The user clicks the link to show the paragraph; then clicks the link again (or clicks the paragraph) to hide it again.

Multiple drop-down paragraphs

To include multiple paragraphs in an expandable section, or to include material in some other format, surround the material to be expanded with **Code** markers in otherwise empty paragraphs; see §7.9.3 [Delimiting expandable drop-down sections](#) on page 228.

Customized drop-down section

To position drop-down links yourself, or to use existing text in your document for the links, or both:

- Place a paragraph in a dedicated format wherever you want a drop-down link; or choose an existing paragraph format (such as a heading or figure title) for this purpose, or even a character format.
- Assign a drop-down link format property to the format, and a corresponding drop-down block format property to the format of material to be expanded; see §7.9.3.1 [Delimiting drop-down links and blocks with paragraph formats](#) on page 228.
- For any existing link text that is *not* in the drop-down link format, surround the link text with **Code** markers that invoke drop-down link macros; do the same for blocks that are not in the drop-down block format, invoking the corresponding drop-down block macros. See §7.9.3.2 [Delimiting drop-down links and blocks with markers](#) on page 229.
- Specify display options for drop-down links and blocks; see §7.9.4 [Configuring drop-down links](#) on page 230 and §7.9.5 [Configuring drop-down blocks](#) on page 233.

7.9.3 Delimiting expandable drop-down sections

Creating drop-down links and expandable blocks involves surrounding material in your document with built-in **Mif2Go** macros for link start, link end, block start, and block end. You can assign these macros as properties of paragraph formats, or you can use **Code** markers to insert the macros, or alternate the use of both methods.

In this section:

§7.9.3.1 [Delimiting drop-down links and blocks with paragraph formats](#) on page 228

§7.9.3.2 [Delimiting drop-down links and blocks with markers](#) on page 229

7.9.3.1 Delimiting drop-down links and blocks with paragraph formats

The following settings use built-in **Mif2Go** macros to surround paragraphs in the designated formats with code to produce the link and expanding block:

```
[HTMLParaStyles]
; Paragraph format = DropDown, DropDownLink, DropDownBlock,
; DropDownStart, or DropDownEnd.
```

or, for the link, to surround character spans:

```
[HTMLCharStyles]
; Character format = DropDownLink, DropDownStart
```

Which format properties to assign depends on the type of link and arrangement of material to be expanded:

- Use `DropDownLink` (for the link) and `DropDownBlock` (for the block) together.

- Use `DropDownStart` (for the link) and `DropDownEnd` (for the block) together.
- Use `DropDown` (for the block) by itself; **Mif2Go** generates the link paragraph.

To expand a single-paragraph block using a **Mif2Go**-inserted link:

```
[HTMLParaStyles]
BlockFormat = DropDown
```

To use an existing paragraph for the link, and a single paragraph for the block:

```
[HTMLParaStyles]
LinkFormat = DropDownLink
BlockFormat = DropDownBlock
```

To make a single- or multiple-paragraph drop-down block start right after the link paragraph:

```
[HTMLParaStyles]
LinkFormat = DropDownStart
BlockEndFormat = DropDownEnd
```

If you use empty paragraphs for *BlockEndFormat*, assign format property *Raw* to the format:

```
[HTMLParaStyles]
BlockEndFormat = DropDownEnd Raw
```

Format property *Raw* is needed *only* if you are using an empty paragraph to delimit the drop-down block.

Table 7-2 shows the effects of these format properties.

Table 7-2 Effects of drop-down format properties

Drop-down style	Format property	Effect
Mif2Go -supplied link, single-paragraph block	<code>DropDown</code>	Treats the current paragraph as the block to be expanded. Places macros <code><\$DropLinkPara></code> and <code><\$DropBlockStart></code> before the paragraph, <code><\$DropBlockEnd></code> after the paragraph.
Variable-text link, single-paragraph block	<code>DropDownLink</code>	Treats the current paragraph or character span as the link. Places macro <code><\$DropLinkStart></code> before the text in the paragraph or character span, <code><\$DropLinkEnd></code> after the text.
	<code>DropDownBlock</code>	Treats the current paragraph as the block to be expanded. Places macro <code><\$DropBlockStart></code> before the paragraph, outside its tags, and <code><\$DropBlockEnd></code> after the closing tags of the paragraph.
Variable-text link, multiple-paragraph block	<code>DropDownStart</code>	Treats the current paragraph as the link, and the next paragraph or paragraphs as the block. Places macro <code><\$DropLinkStart></code> before the text in the current paragraph, <code><\$DropLinkEnd></code> after the text, then <code><\$DropBlockStart></code> after the paragraph.
	<code>DropDownEnd</code>	Treats the current paragraph as the last paragraph in the block to be expanded. Places macro <code><\$DropBlockEnd></code> after the current paragraph.

7.9.3.2 Delimiting drop-down links and blocks with markers

You can use **Code** markers to surround drop-down links and expandable blocks with built-in **Mif2Go** macro code.

To delimit links and blocks with **Code** markers, place a **Code** marker in each of the following places, with the indicated content:

- At the beginning of the link paragraph before any text, with content that depends on the type of drop-down link:

<u>DropLinkType</u>	<u>Starting Code marker content</u>
Icon	<\$DropLinkStart><\$DropOpenIcon><\$DropCloseIcon>
Button	<\$DropLinkStart><\$DropButton>
Text	<\$DropLinkStart>

(See §7.9.4.1 [Specifying the type of link for drop-down sections](#) on page 230.)

- At the end of the link paragraph, after any text:
<\$DropLinkEnd>
- In a dedicated paragraph before the block:
<\$DropBlockStart>
- In a dedicated paragraph after the block:
<\$DropBlockEnd>

The macros for a drop-down block must be outside any paragraph tags. Put the **Code** markers in otherwise empty paragraphs of their own, just before the first block paragraph and just after the last block paragraph. Use a dedicated paragraph format for the markers, and assign the following property to the format:

```
[HTMLParaStyles]
CodeMarkerFmt = Raw
```

See §21.3.6 [Stripping paragraph properties](#) on page 650.

7.9.4 Configuring drop-down links

By default, **Mif2Go** uses icons for the drop-down link for an expandable section. Optionally, an icon can be followed by text: either fixed text, or an existing paragraph or character span in your document. Instead of icons or icons plus text, you can use text only, or buttons.

In this section:

- §7.9.4.1 [Specifying the type of link for drop-down sections](#) on page 230
- §7.9.4.2 [Configuring icons for drop-down links](#) on page 231
- §7.9.4.3 [Configuring buttons for drop-down links](#) on page 232
- §7.9.4.4 [Configuring text for drop-down links](#) on page 232
- §7.9.4.5 [Modifying code for drop-down links](#) on page 232

7.9.4.1 Specifying the type of link for drop-down sections

To specify the type of drop-down link to use for an expandable section:

```
[DropDowns]
; DropLinkType = Icon (default, optional text), Button, or Text (only)
DropLinkType = Icon
```

When DropLinkType=Icon, **Mif2Go** inserts an icon at the start of each drop-down link paragraph. You can specify the graphics to use for icons, and you can modify the alt text; see §7.9.4.2 [Configuring icons for drop-down links](#) on page 231.

When DropLinkType=Button, **Mif2Go** inserts a button in each drop-down link paragraph. You can specify the label on the button; see §7.9.4.3 [Configuring buttons for drop-down links](#) on page 232.

When `DropLinkType=Text`, the link consists only of text: either default text, or the text of a paragraph or character span in your document; see §7.9.4.4 [Configuring text for drop-down links](#) on page 232.

Regardless of link type, **Mif2Go** inserts icon, button, or default text ahead of whatever text is already present in each drop-down link paragraph. If you have not designated a drop-down link paragraph, **Mif2Go** creates a paragraph for the link, just before each drop-down block.

By default, **Mif2Go** includes JavaScript code that works for all types of drop-down links. However, you can choose to have **Mif2Go** include JavaScript code only for the type of link you specify via `DropLinkType`; this results in slightly less JavaScript code.

To restrict JavaScript drop-down code to the link type specified by `DropLinkType`:

```
[DropDowns]
; UseCompositeDropJS = Yes (default, use JS that works for all
; settings of DropLinkType), or No (use JS specific to current
; DropLinkType setting, slightly less JS code)
UseCompositeDropJS = No
```

When `UseCompositeDropJS=No`, all drop-down sections in a given `FrameMaker` file must use the same type of drop-down link, because the type determines the JavaScript that is included or referenced in the output. See §7.9.7 [Deploying JavaScript code for drop-down sections](#) on page 234.

7.9.4.2 Configuring icons for drop-down links

When `DropLinkType=Icon`, each drop-down link starts with an icon. If you provide text content for the link paragraph or character span, the text follows the icon.

Mif2Go can provide a default icon pair:

<u>Icon</u>	<u>Icon graphic file</u>	<u>Alternate text</u>
	dropopen.gif	Click to open.
	dropclose.gif	Click to close.

To have **Mif2Go** write these default icon files to the project directory:

```
[DropDowns]
; WriteDropIconFiles = No (default) or Yes (write to project
; directory)
WriteDropIconFiles = Yes
```

When `WriteDropIconFiles=Yes`, **Mif2Go** creates default drop-down icons in the project directory. When `WriteDropIconFiles=No`, you must provide the icon files. If you want the icons in a directory other than the project directory, perhaps with other graphics, you must place the icon files there yourself, and specify a relative path to their location.

To rename or relocate drop-down icon files:

```
[DropDowns]
DropOpenIconFile = path/to/dropopen.gif
DropCloseIconFile = path/to/dropclose.gif
```

The default location is the project directory. If you specify a relative path, it is relative to the project directory. Do not use an absolute path.

To specify different alt text for the icons:

```
[DropDowns]
DropOpenIconAlt = Click to open.
DropCloseIconAlt = Click to close.
```

To change the macro code that displays icons, see §7.9.4.5 [Modifying code for drop-down links](#) on page 232.

7.9.4.3 Configuring buttons for drop-down links

When DropLinkType=Button, each drop-down link consists of a button, with a text label on the button itself. The default label text is **More** when the drop-down section is closed, **Less** when the section is open.

To change the default button labels:

```
[DropDowns]
DropButtonOpenLabel = More
DropButtonCloseLabel = Less
```

To change the macro code that displays buttons, see §7.9.4.5 [Modifying code for drop-down links](#) on page 232.

7.9.4.4 Configuring text for drop-down links

When DropLinkType=Text, the link consists only of text: either fixed text, or existing text in a paragraph or character span in your document.

Fixed text For fixed text, by default **Mif2Go** inserts a paragraph with content “**Click here.**” as the link. To specify different fixed text for the link:

```
[DropDowns]
DropText = Click here.
```

Existing text To use existing text for the link, delimit the text with link-start and link-end macros; see §7.9.3 [Delimiting expandable drop-down sections](#) on page 228.

To change the macro code that displays fixed text, see §7.9.4.5 [Modifying code for drop-down links](#) on page 232.

7.9.4.5 Modifying code for drop-down links

You can redefine any of the built-in drop-down link macros by changing the default code that is assigned to the macro name. Each macro name serves as a keyword in your project configuration file; you change the definition by assigning replacement code to the macro name. When you assign code to a macro name in the configuration file, the entire setting must be all on one line, even if it does not look that way here.

Ordinarily you should not need to include any of the settings described in this section.

Link ID prefix Because an ID used in JavaScript must start with a letter, by default **Mif2Go** prefixes the incremental value of the drop-down link ID with drop. To change the drop-down link ID prefix:

```
[DropDowns]
DropIDPrefix = drop
```

Link macros To change the code for creating drop-down links, include the following settings to redefine the built-in macros:

```
[DropDowns]
; Default macros for link start/end:
DropLinkStart = <a class="<$DropClass>"\n <$DropLinkAttr>>
DropLinkAttr = href="javascript:doSection('<$$_DropID>');void 0;"
DropLinkEnd = </a>
```

If you use DropLinkStart, include CSS for the text to match its content; also see: §7.9.6 [Providing CSS for drop-down links and blocks](#) on page 233:

```
[DropDowns]
DropLinkParaStart = <p class="<$DropClass>">
DropLinkParaText = <$DropText>
DropLinkParaEnd = </p>
```

<\$DropLinkStart> follows <\$DropLinkParaStart>. If DropLinkType=Icon, both icons follow <\$DropLinkStart>. Unless DropLinkType=Button, next come <\$DropLinkParaText> then <\$DropLinkEnd>.

Icon macros To change the code for drop-down icons:

```
[DropDowns]
DropOpenIcon = <img\n src="<$DropOpenIconFile>" id="io<$$_DropID>"
style="border:0;" alt="<$DropOpenIconAlt>">
DropCloseIcon = <img\n src="<$DropCloseIconFile>" id="ic<$$_DropID>"
style="display:none;border:0;" alt="">\n
```

Button macros To change the code for drop-down buttons:

```
[DropDowns]
DropButton = \n<button type="button" class="<$DropClass>" id=
"bu<$$_DropID>" <$DropButtonAttr>><$DropButtonOpenLabel></button>\n
DropButtonAttr = onclick="doSection('<$$_DropID>')"
```

Text macro To change the code for fixed-text links:

```
[DropDowns]
DropLinkPara = <p class="<$DropClass>">
<$DropLinkStart><$DropText><$DropLinkEnd></p>
```

7.9.5 Configuring drop-down blocks

To specify whether clicking inside an open drop-down block should close the block:

```
[DropDowns]
; ClickBlockToClose = Yes (default)
; or No (use if any links inside block)
ClickBlockToClose = Yes
```

If any of your drop-down blocks contain links, set ClickBlockToClose=No.

To change the code for creating drop-down blocks, include the following settings to redefine the built-in macros:

```
[DropDowns]
DropBlockStart = <div class="<$DropClass>" id="<$$_DropID>"
style="display:none;" <$DropDivAttr>>\n
; If ClickBlockToClose=No, this is omitted:
DropDivAttr = onclick="noSection('<$$_DropID>')'"
DropBlockEnd = </div>\n
```

Each setting must be all on one line in your configuration file, even if it does not look that way here. *Ordinarily you should not need to include these settings.*

7.9.6 Providing CSS for drop-down links and blocks

By default, the CSS class for links and blocks is dropdown. To specify a different class:

```
[DropDowns]
DropClass = dropdown
```

The same class name can serve for all link types, and also for blocks. Use CSS to differentiate:

```
p.dropdown { drop-link text stuff }
a.dropdown { drop-link icon stuff }
button.dropdown { drop-link button stuff }
div.dropdown { drop block stuff }
```


See §22 [Setting up CSS for HTML](#) on page 681.

7.9.7 Deploying JavaScript code for drop-down sections

Based on the configuration settings you specify for drop-down links and blocks, **Mif2Go** creates a macro that contains the JavaScript code for the drop-down sections. You can modify this code, and rename, relocate, or replace the code.

In this section:

§7.9.7.1 [Naming the JavaScript macro for drop-down sections](#) on page 234

§7.9.7.2 [Locating JavaScript code for drop-down sections](#) on page 234

§7.9.7.3 [Directing Mif2Go to write drop-down JavaScript code](#) on page 235

§7.9.7.4 [Inspecting the JavaScript code for drop-down sections](#) on page 235

7.9.7.1 Naming the JavaScript macro for drop-down sections

By default, the name of the JavaScript macro is `$DropJS`. To specify a different name for this macro, and by implication, to supply your own macro body:

```
[DropDowns]
; This is the code that goes in the <head> or in a JS file:
DropJSCode = <$DropJS>
```

Enclose the macro name in angle brackets. Including this setting is tantamount to saying:

Do not write this macro; I am supplying my own version.

If you specify a value for `DropJSCode`, you must provide the named macro in your configuration file or in a macro library.

If you do not specify a value for `DropJSCode`, or if you do not provide the named macro, **Mif2Go** includes either the composite JavaScript code (see §7.9.4.1 [Specifying the type of link for drop-down sections](#) on page 230) or one of four built-in versions of this macro; see §7.9.7.4 [Inspecting the JavaScript code for drop-down sections](#) on page 235.

7.9.7.2 Locating JavaScript code for drop-down sections

By default, for most output types **Mif2Go** inserts JavaScript code for drop-down sections in the `<head>` section of each HTML file that contains one or more drop-down sections. For OmniHelp, **Mif2Go** includes the JavaScript code in viewer files `ohctrl.js` and `ohmain.js`. For any output type, you can direct **Mif2Go** to reference the code in a separate JavaScript library instead.

To specify where the JavaScript code resides:

```
[DropDowns]
; DropJSLocation = Head (to insert the code in <script> tags),
; None (if the code is included elsewhere, as for OmniHelp),
; or a filename to reference in a JS link in the <head>.
DropJSLocation = Head
```

When `DropJSLocation=Head`, **Mif2Go** places JavaScript code in the `<head>` section of each output HTML file that includes at least one drop-down section:

```
<script language="JavaScript" type="text/javascript">
<!--
<$DropJS>
//-->
</script>
```

Macro `$DropJS` is expanded when **Mif2Go** writes the output HTML file.

When `DropJSLocation=None`, **Mif2Go** assumes you are supplying a JavaScript library for which a reference already exists, possibly configured as part of a value for `Head` in the `[Inserts]` section. See §28.9.2 [Invoking macros at predetermined points in output](#) on page 821.

When `DropJSLocation=filename`, **Mif2Go** places the following reference in the `<head>` section:

```
<script language="JavaScript" type="text/javascript"
src="<$DropJSLocation>"></script>
```

Macro `$DropJSLocation` is expanded when **Mif2Go** writes the output HTML file. The file specification you provide for `filename` can include a path relative to the project directory. Although you can specify an absolute path, we advise against it. Also, a path that includes a drive specification will not work.

7.9.7.3 Directing Mif2Go to write drop-down JavaScript code

By default, **Mif2Go** does not create an external version of the drop-down JavaScript code. To have **Mif2Go** write the JavaScript code to a file in the project directory, when you specify a file for `DropJSLocation`:

```
[DropDowns]
; WriteDropJSFile = No (default, you provide it) or Yes
WriteDropJSFile = Yes
```

When `WriteDropJSFile=No`, **Mif2Go** assumes that the file you specified for `DropJSLocation` is an existing JavaScript library that you do not want overwritten.

When `WriteDropJSFile=Yes`, **Mif2Go** overwrites the file you specified for `DropJSLocation` if it is in the project directory, or creates the file if it does not already exist in the project directory.

`WriteDropJSFile` takes effect only when `DropJSLocation=filename` (see §7.9.7.2 [Locating JavaScript code for drop-down sections](#) on page 234). And while any path information included in `filename` is used in the link in the `<head>` section, **Mif2Go** writes the file itself to the project directory, for security reasons. This means that if `DropJSLocation` specifies a location *other than the project directory*, you must move the file to the other directory, with a `SystemEndCommand` setting; see §34.4.1 [Specifying system commands](#) on page 938.

7.9.7.4 Inspecting the JavaScript code for drop-down sections

The JavaScript functions included in macro `$DropJS` differ according to whether `UseCompositeDropJS=Yes` or `No`; and if `No`, according to the link type.

In this section:

§7.9.7.4.1 [JavaScript code when UseCompositeDropJS=Yes](#) on page 235

§7.9.7.4.2 [JavaScript code when DropLinkType=Icon](#) on page 236

§7.9.7.4.3 [JavaScript code when DropLinkType=Button](#) on page 237

§7.9.7.4.4 [JavaScript code when DropLinkType=Text](#) on page 237

See also:

§7.9.4.1 [Specifying the type of link for drop-down sections](#) on page 230.

7.9.7.4.1 JavaScript code when UseCompositeDropJS=Yes

```
[DropJS]
function doSection(id){
  var but = document.getElementById("bu" + id)
```

```

var imop = document.getElementById("io" + id)
var imcl = document.getElementById("ic" + id)
var idiv = document.getElementById(id)
if (idiv.style.display=="none") {
  idiv.style.display=""
  if (but != null)
    but.innerHTML("<$DropButtonCloseLabel>"
  if (imop != null)
    imop.style.display="none"
  if (imcl != null)
    imcl.style.display=""
} else {
  idiv.style.display="none"
  if (but != null)
    but.innerHTML("<$DropButtonOpenLabel>"
  if (imop != null)
    imop.style.display=""
  if (imcl != null)
    imcl.style.display="none"
}
return false;
}
function noSection(id){
  var but = document.getElementById("bu" + id)
  var imop = document.getElementById("io" + id)
  var imcl = document.getElementById("ic" + id)
  var idiv = document.getElementById(id)
  if (idiv.style.display=="") {
    idiv.style.display="none"
    if (but != null)
      but.innerHTML("<$DropButtonOpenLabel>"
    if (imop != null)
      imop.style.display=""
    if (imcl != null)
      imcl.style.display="none"
  }
}

```

7.9.7.4.2 JavaScript code when DropLinkType=Icon

```

[DropJS]
function doSection(id){
  var imop = document.getElementById("io" + id)
  var imcl = document.getElementById("ic" + id)
  var idiv = document.getElementById(id)
  if (idiv.style.display=="none") {
    idiv.style.display=""
    imop.style.display="none"
    imcl.style.display=""
  } else {
    idiv.style.display="none"
    imop.style.display=""
    imcl.style.display="none"
  }
  return false;
}
function noSection(id){
  var imop = document.getElementById("io" + id)
  var imcl = document.getElementById("ic" + id)
  var idiv = document.getElementById(id)
  if (idiv.style.display=="") {
    idiv.style.display="none"
    imop.style.display=""
  }
}

```

```

        imcl.style.display="none"
    }
}

```

7.9.7.4.3 JavaScript code when DropLinkType=Button

```

[DropJS]
function doSection(id){
    var but = document.getElementById("bu" + id)
    var idiv = document.getElementById(id)
    if (idiv.style.display=="none") {
        idiv.style.display=""
        but.innerHTML("<$DropButtonCloseLabel>"
    } else {
        idiv.style.display="none"
        but.innerHTML("<$DropButtonOpenLabel>"
    }
    return false;
}
function noSection(id){
    var but = document.getElementById("bu" + id)
    var idiv = document.getElementById(id)
    if (idiv.style.display=="") {
        idiv.style.display="none"
        but.innerHTML("<$DropButtonOpenLabel>"
    }
}

```

7.9.7.4.4 JavaScript code when DropLinkType=Text

```

[DropJS]
function doSection(id){
    var idiv = document.getElementById(id)
    if (idiv.style.display=="none") {
        idiv.style.display=""
    } else {
        idiv.style.display="none"
    }
    return false;
}
function noSection(id){
    var idiv = document.getElementById(id)
    if (idiv.style.display=="") {
        idiv.style.display="none"
    }
}

```

7.9.8 Emulating Web Works Publisher drop-down hotspots

To create expandable drop-down sections in Web Works Publisher, you map a paragraph that you want to be expandable to one of two WWP styles:

- *DropDownClosed* to make the hotspot closed by default
- *DropDownOpen* to make the hotspot open by default.

Everything following the hotspot paragraph is included in the drop-down content up to the next paragraph mapped to *DropDownClosed*, *DropDownOpen*, or one of the standard WWP heading styles. You can emulate this method using **Mif2Go** macros, either with or without built-in **Mif2Go** drop-down controls. Jim Owens has kindly allowed us to present the macros he developed for this purpose.

In this section:

§7.9.8.1 [Creating drop-down hotspots with Mif2Go controls and macros](#) on page 238

§7.9.8.2 [Creating drop-down hotspots with Mif2Go macros only](#) on page 238

7.9.8.1 Creating drop-down hotspots with Mif2Go controls and macros

The following settings use **Mif2Go** format property DropDownLink and two macros to handle open/close actions for dedicated drop-down paragraph format *DropPara*.

```
[HTMLParaStyles]
DropPara = DropDownLink CodeBefore CodeAfter

[ParaStyleCodeBefore]
; At the start of any of the following paragraphs,
; close any open drop-down blocks:
DropPara = <$DropDownBlockClose>
; List any other paragraphs that should end a drop-down block:
H1 = <$DropDownBlockClose>
H2 = <$DropDownBlockClose>
H3 = <$DropDownBlockClose>
H4 = <$DropDownBlockClose>
H5 = <$DropDownBlockClose>

[ParaStyleCodeAfter]
DropPara = <$DropDownBlockOpen>

[Inserts]
; At end of body, close any open drop-down blocks:
Bottom = <$DropDownBlockClose>

[DropDownBlockOpen]
; After DropPara, insert javascript to open a new drop-down block,
; and set a flag to signify that the block is open. The javascript
; includes a counter to identify the drop-down section:
<$$DropDownCount++>
; Strip leading zeroes:
<div class="dropdown" id="drop<$$DropDownCount as %0.1d>"
style="display:none;"
onclick="noSection('drop<$$DropDownCount as %0.1d>')">
<$$Flag_DropDownBlockOpen = 1>

[DropDownBlockClose]
; Before DropPara or H1 through H5 or </body>,
; check a flag to see if a drop-down block is open;
; if so, close the drop-down block and clear the flag:
<$_if ($$Flag_DropDownBlockOpen)>
</div>
<$$Flag_DropDownBlockOpen = 0>
<$_endif>

[MacroVariables]
; Put any macro definition sections before this section.
Flag_DropDownBlockOpen = 0
DropDownCount = 0
```

7.9.8.2 Creating drop-down hotspots with Mif2Go macros only

The following settings use three macros to handle open/close actions for dedicated drop-down paragraph format *DropPara*.

```
[HTMLParaStyles]
DropPara = CodeBefore CodeStart CodeAfter

[ParaStyleCodeBefore]
; At the start of any of the following paragraphs,
```

```

; close any open drop-down blocks:
DropPara = <$DropDownBlockClose>
; List any other paragraphs that should end a drop-down block:
H1 = <$DropDownBlockClose>
H2 = <$DropDownBlockClose>
H3 = <$DropDownBlockClose>
H4 = <$DropDownBlockClose>
H5 = <$DropDownBlockClose>

[ParaStyleCodeStart]
; Before the DropPara text, insert a drop-down link:
DropPara = <$DropDownLinkOpen>
[ParaStyleCodeAfter]
DropPara = <$DropDownBlockOpen>

[Inserts]
; At end of body, close any open drop-down blocks:
Bottom = <$DropDownBlockClose>
[DropDownLinkOpen]
; Before DropPara text, set a new drop-down link:
<$$DropDownCount++>
<a class="dropdown"
  href="javascript:doSection('drop<$$DropDownCount>');void 0;">
"
  style="border:0;" alt="Click to open.">
"
  style="display:none;border:0;" alt="Click to close.">
[DropDownBlockOpen]
; After DropPara, insert javascript to open a new drop-down block,
; and set a flag to signify that the block is open. The javascript
; includes a counter to identify the drop-down section:
<div class="dropdown" id="drop<$$DropDownCount>" style="display:none;"
onclick="noSection('drop<$$DropDownCount>')">
<$$Flag_DropDownBlockOpen = 1>

[DropDownBlockClose]
; Before DropPara or H1 through H5 or </body>,
; check a flag to see if a drop-down block is open;
; if so, close the drop-down block and clear the flag:
<$_if ($$Flag_DropDownBlockOpen)>
</div>
<$$Flag_DropDownBlockOpen = 0>
<$DropDownLinkOpen>
<$_endif>

[MacroVariables]
; Put any macro definition sections before this section.
Flag_DropDownBlockOpen = 0
DropDownCount = 0

```

7.10 Setting up Context Sensitive Help (CSH)

Creating a link from an application program to a topic in your Help file is pretty much the same process for all flavors of WinHelp and HTML-based Help. Differences among the tools used to develop the application dictate minor differences in the process.

In this section:

§7.10.1 [Understanding how CSH works](#) on page 240

§7.10.2 [Specifying CSH mappings](#) on page 241

See also:

§9.12 [Setting up CSH for HTML Help](#) on page 326

§10.11 [Setting up CSH for OmniHelp](#) on page 364

§11.12 [Setting up CSH for JavaHelp or Oracle Help](#) on page 401

7.10.1 Understanding how CSH works

When an application program calls on a Help system to display a topic, the program might pass a *number* to the Help system to identify the requested topic; this is usually the case for HTML Help, and always for WinHelp. The Help system, however, uses a *name* to identify a topic; in fact, a topic can have any number of names. In the application for which you are creating CSH, each click of a **Help** button calls a number (a *numeric ID*), and each number has to be mapped to a Help topic name (a *symbolic ID*). A *map file* provides the necessary links from program to Help system; for some Help systems, an *alias file* associates each symbolic ID with the name of the Help file:

<u>Program</u>		<u>Map file</u>		<u>Help file</u>
Numeric ID	>	Symbolic ID	>	Help topic

For HTML Help, the program can pass a file name instead of a number, eliminating the need for map and alias files, but this is rarely done. For JavaHelp, Oracle Help for Java, and OmniHelp, the program can pass a name instead of a number.

Numeric ID A *numeric ID* is usually an integer. You might specify the numeric IDs for the developer of the application program, or the developer might specify them; which way depends on development tools and project work flow. If the application program is developed in Visual Basic, the developer enters the numbers on a form; in Visual C/C++, what the developer does depends on which API call variant is in use.

Symbolic ID A *symbolic ID* consists of the following:

- a special prefix, either `IDH_` or `HIDC_`, that identifies the symbolic ID as a CSH destination
- a name, usually furnished by the application developer, for the application feature involved (such as a button, dialog, or text box).

Each symbolic ID must be unique in your Help project.

Compilers have built-in support for `IDH_`, so use that prefix if possible. If the developers are using Microsoft Foundation Classes, `HIDC_` works also. For HTML Help and OmniHelp, you specify in the configuration file which prefix(es) you are using. WinHelp also uses prefixes, `IDH_` in particular; and reports any such entries in your map file for which you did not provide a destination in a topic.

Map file A *map file* is an ASCII file that contains a line for each link from program to Help system. In some programming environments, such as Visual C/C++, this file is produced for you; in others, such as Visual Basic, you create the map file yourself. For C or C++ the map file is usually named `resource.h`. For JavaHelp and Oracle Help for Java, **Mif2Go** creates the map file (with extension `.jhm`), and writes the symbolic IDs to the file. No map file is needed for OmniHelp.

Alias file An *alias file* is an ASCII file that contains a line for each symbolic ID, associating that ID with the name of the Help file that contains the relevant CSH destination. For HTML Help and OmniHelp, you identify each symbolic ID as an *alias*, which gets listed in the alias file; **Mif2Go** can generate the alias file for you. WinHelp links automatically, without an alias file; no additional author actions are required. Help Workshop provides the prefixes.

7.10.2 Specifying CSH mappings

To provide CSH when **Mif2Go** generates Help files for your project:

1. **Give each target topic a **TopicAlias** marker that contains a symbolic ID.**
In your FrameMaker document, insert a **TopicAlias** marker in each topic that will be the target of a call from the application program. The content of the marker is the symbolic ID for the topic. Insert a separate **TopicAlias** marker with a unique symbolic ID for each call from the application. Put the marker at the start of the text you want displayed. To insert a **TopicAlias** marker, see §34.1.2 [Using markers to add links and instructions](#) on page 935.
2. **Create or obtain a map file** (possibly except JavaHelp and Oracle Help for Java; see §11.12 [Setting up CSH for JavaHelp or Oracle Help](#) on page 401).
3. **Specify prefixes that identify CSH links** (HTML Help or OmniHelp).
List topic-name prefixes in the configuration file, to identify **TopicAlias** markers intended for CSH use. If you do not specify any prefixes, all **TopicAlias** markers are included.
4. **Map the appropriate application-provided number to each symbolic ID.**
For C/C++ applications, usually the developer provides a map file. If not, for WinHelp or HTML Help you can use a simple syntax described in the Help provided for those Help systems. Otherwise, for each Help call in the program, add a line of the following form to the map file:

```
#define symbolic_ID numeric_ID
```

You cannot map multiple numeric IDs to the same symbolic ID; each entry in the map file must specify a different symbolic ID. If you need CSH links to the same Help topic from more than one point in the application, include in the topic a separate **TopicAlias** marker with a unique symbolic ID for each such Help call.

5. **Add a map-file entry to the Help project file** (WinHelp or HTML Help).
In the [MAP] section of the Help project file (*MyDoc.hpj* or *MyDoc.hhp*), add a line of the following form to identify the map file:

```
#include MapFileName.h
```

Mif2Go creates a CSH link destination from each **TopicAlias** marker whose name starts with one of the prefixes you specified in [Step 3](#), or all the **TopicAlias** markers if you did not specify any prefixes. Make sure the symbolic IDs in the **TopicAlias** markers are spelled the same way as in the map file.

By default, **Mif2Go** removes punctuation and spaces from the **TopicAlias** marker content. If your HTML-based Help system requires CSH IDs that use characters such as periods, set the following option:

```
[HTMLOptions]
; UseRawNewlinks = No (default, remove punctuation, spaces)
; or Yes (as is)
UseRawNewlinks=Yes
```

7.11 Setting up a dynamic modular Help system

Suppose you are providing Help for a product with several optional modules, and you want to supply each customer with information only about the modules licensed by that customer. Or, suppose a user decides not to install a module, or adds a new module later. Or, suppose you have created reusable Help modules that can be incorporated into any of a number of main Help systems, such as “Help on Help”.

Any Help system that **Mif2Go** generates can be made to access more than one module, yet appear to the user as a single unit. The user accesses a single Help file, and sees integrated contents, index, and related-topic links, containing information only for the relevant modules. Additional modules are loaded dynamically, when a user clicks a **Contents**, **Index**, **Related Topics**, or **Search** entry that references a separate module.

Methods and configuration settings vary according to which type of Help system you are generating. See the following for more information:

<u>Help system</u>	<u>Reference</u>
WinHelp	§8.2.11 Providing multiple .hlp files on page 249
MS HTML Help	§9.15 Mapping and merging CHM files on page 336
OmniHelp	§10.12 Merging OmniHelp projects on page 366
JavaHelp, Oracle Help for Java	§11.11 Merging JavaHelp or Oracle Help systems on page 400
Eclipse Help	§12.6 Merging Eclipse Help projects on page 415

8 Generating WinHelp

Mif2Go produces RTF topic files, CNT (contents) files, and WMF graphics files for WinHelp. Most format conversion options are the same as for print RTF. This section addresses issues that are specific to WinHelp. Topics covered:

- §8.1 [Obtaining tools for WinHelp](#) on page 243
- §8.2 [Setting up a WinHelp project](#) on page 243
- §8.3 [Converting text](#) on page 252
- §8.4 [Converting cross references](#) on page 259
- §8.5 [Converting tables to WinHelp RTF](#) on page 261
- §8.6 [Managing graphics for WinHelp](#) on page 263
- §8.7 [Converting generated files for WinHelp](#) on page 265
- §8.8 [Configuring WinHelp topics](#) on page 267
- §8.9 [Creating jumps and pop-ups for WinHelp](#) on page 272
- §8.10 [Invoking WinHelp macros](#) on page 284
- §8.11 [Creating related-topic links in WinHelp](#) on page 285
- §8.12 [Configuring index entries for WinHelp](#) on page 287
- §8.13 [Configuring contents for WinHelp](#) on page 288
- §8.14 [Creating browse sequences](#) on page 292

See also:

- §6 [Converting to print RTF](#) on page 141, for information about settings that work the same way for print RTF and for WinHelp.
- §7 [Producing on-line Help](#) on page 199, for information about configuring contents and index, providing related-topics links, supporting context-sensitive help, and merging help projects.

8.1 Obtaining tools for WinHelp

To generate WinHelp you need Microsoft Help Workshop, `hcx.exe`. However, this program is no longer available. If you have Microsoft Visual Studio 2008, Help Workshop was reportedly included in that version. If you do not have access to Help Workshop, you must choose a Help output type other than WinHelp; see §7.1 [Weighing Help-system alternatives](#) on page 199.

To view WinHelp, users with systems running Windows Vista or Windows 7 will have to download a new WinHelp engine from Microsoft:

- Windows Vista: <http://tinyurl.com/4pkahu2>
- Windows 7: <http://tinyurl.com/67qt76f>

Note: The WinHelp engine, `winhlp32.exe`, cannot be distributed by third parties, so do not include it when you distribute a WinHelp system. Every installation of `winhlp32.exe` requires Microsoft validation.

8.2 Setting up a WinHelp project

When you specify WinHelp for output, **Mif2Go** produces the `.hpx`, the `.cnt`, the `.wmf` graphics, and the WinHelp-coded `.rtf` files. You open the `.hpx` in Help Workshop, and click **Compile**. You can view the result in Help Workshop.

In this section:

- §8.2.1 [Setting up a WinHelp project](#) on page 244
- §8.2.2 [Choosing set-up options for a WinHelp project](#) on page 244
- §8.2.3 [Deciding where to locate configuration settings](#) on page 245
- §8.2.5 [Understanding initial set-up requirements](#) on page 246
- §8.2.6 [Deciding whether to regenerate the WinHelp project file](#) on page 246
- §8.2.7 [Accommodating platform differences](#) on page 247
- §8.2.8 [Setting basic WinHelp options in the configuration file](#) on page 248
- §8.2.9 [Including ObjectIDs in WinHelp](#) on page 249
- §8.2.10 [Handling page breaks and section breaks](#) on page 249
- §8.2.11 [Providing multiple .hlp files](#) on page 249
- §8.2.12 [Integrating WinHelp from RoboHelp](#) on page 250
- §8.2.13 [Compiling a WinHelp project](#) on page 250
- §8.2.14 [Checking WinHelp RTF files for Mif2Go version](#) on page 251

8.2.1 Setting up a WinHelp project

To set up a WinHelp project:

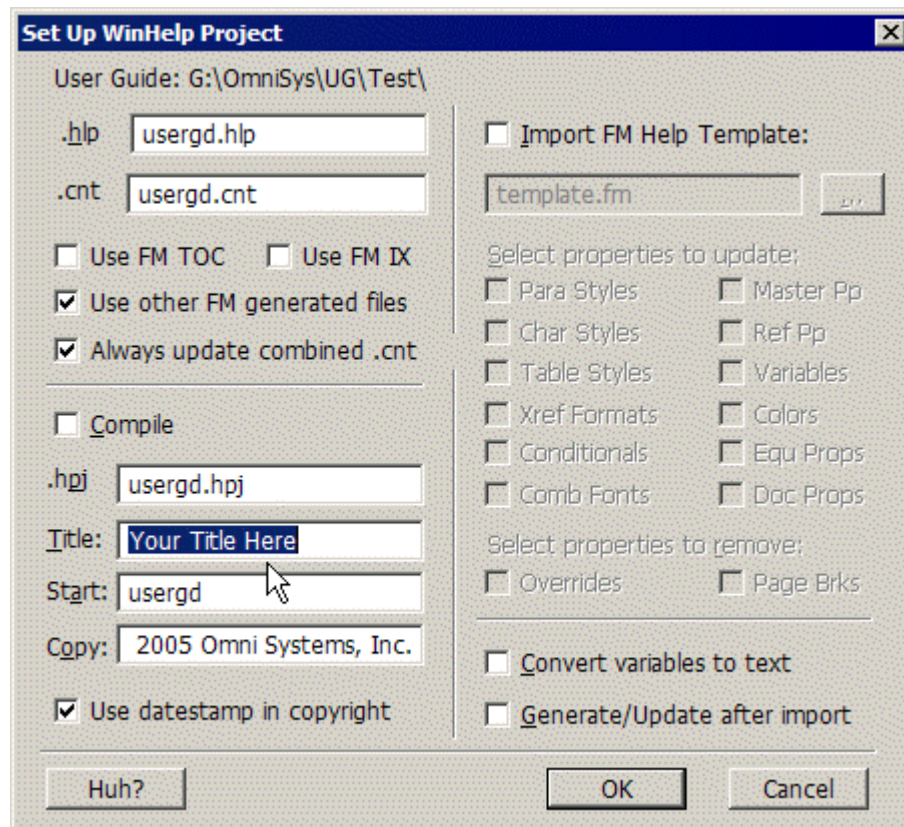
1. Create a project directory for WinHelp RTF files, separate from the directory where your FrameMaker document is located.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens.
3. Name your WinHelp project, and browse to the project directory you created in [Step 1](#) (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
4. In the *Choose Project* dialog (see §3.3 [Creating a Mif2Go conversion project](#) on page 78), choose WinHelp 4/95 RTF.
5. Select options in the *Set Up WinHelp Project* dialog (see §8.2.2 [Choosing set-up options for a WinHelp project](#) on page 244).
6. Dismiss the *Conversion Designer* dialog.
7. Use a text editor to specify settings in configuration file `_m2winhelp.ini` (see §4.1 [Working with Mif2Go configuration files](#) on page 91).
8. **Important:** To make the configuration fit your usage of headings to start topics, pay special attention to sections `[HelpContentsLevels]` (see §7.2.1 [Checking automatic Help topic assignments](#) on page 203) and `[HelpStyles]` (see §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268).

8.2.2 Choosing set-up options for a WinHelp project

When you select WinHelp as the output type for a new project, the *Set Up* dialog shown in [Figure 8-1](#) opens. [Table 8-1](#) shows the corresponding settings in the configuration file.

See also:

- §7 [Producing on-line Help](#) on page 199

Figure 8-1 Set Up WinHelp Project**Table 8-1 WinHelp set-up options and configuration settings**

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
.hlp file name	[HelpContents]	CntBase= <i>myfile</i> .hlp	<i>FMbook</i> .hlp	8.13.1
.cnt file name	[HelpContents]	CntName= <i>myfile</i> .cnt	<i>FMbook</i> .cnt	8.13.1
Always update combined .cnt	[HelpOptions]	MakeCombinedCnt=Yes	Yes	8.2.8
Compile:	[Automation]	CompileHelp=Yes	No	8.2.13
.hpi file name	[HelpOptions]	HPJFileName= <i>myproj</i> .hpi	<i>FMbook</i> .hpi	8.2.13
Title	[HelpContents]	CntTitle= <i>My Project</i>	Your Title Here	8.13.1
Starting topic file	[HelpContents]	CntTopic= <i>myfile</i>	Book or file name	8.13.1
Copyright	[HelpOptions]	HelpCopyright= <i>statement</i>	No default	See
Use datestamp in copyright	[HelpOptions]	HelpCopyDate=%date	%date	.hpi file

8.2.3 Deciding where to locate configuration settings

When you set up a WinHelp project from within FrameMaker, if configuration file `_m2winhelp.ini` is not already present in the project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

Which configuration file? To configure WinHelp output, add settings to one of the following files, depending on the desired scope of each setting:

<u>Scope</u>	<u>Configuration file</u>	<u>Location</u>
Current project only	_m2winhelp.ini	Current project directory
All WinHelp projects	local_m2winhelp_config.ini	%omsyshome%\m2g\local\config\projects

See §30.5 [Deciding which configuration file to edit](#) on page 856.

8.2.4 Preparing a document for conversion to WinHelp

To generate WinHelp from a FrameMaker document, observe the following guidelines:

- Keep file names short; very long names might exceed limits in WinHelp, especially the 64-character limit on lines in the .cnt file. See §2.2 [Naming FrameMaker formats](#) on page 66 for additional file-name restrictions.
- Provide a conversion template that reduces the size range of headings, and eliminates page numbers from cross references; see §2.4 [Importing formats from a conversion template](#) on page 67.
- Avoid using tabs. FrameMaker uses a unique tabbing scheme, to eliminate the dependency on font metrics that is present in Word and in WinHelp.
- Provide versions of graphics in BMP or WMF format; see §5.7 [Processing graphics](#) on page 126 for more information.

8.2.5 Understanding initial set-up requirements

To set up a WinHelp project and have **Mif2Go** create a WinHelp project file (.hbj), you must do the following:

- make sure no m2winhelp.ini configuration file is present in the project directory
- run **Mif2Go** from within FrameMaker.

If you run **Mif2Go** from the command line, or if a configuration file is already present, no .hbj file is created.

If you are creating a multi-file WinHelp project, always work from a FrameMaker book file rather than from a single document file. If necessary, create a new book containing just the files you want to include in WinHelp. Or, use the existing book file, if the only difference from a print version of the book is that you are excluding the index.

To compile WinHelp, you will need Microsoft Help Workshop; see §1.3.5 [Obtain tools for Help systems or eBooks](#) on page 58.

8.2.6 Deciding whether to regenerate the WinHelp project file

When you use **Mif2Go** to generate WinHelp from within FrameMaker, **Mif2Go** writes an .hbj project file during set-up, and rewrites it later only under certain conditions.

To specify whether **Mif2Go** should generate the .hbj project file anew each time you run the conversion:

```
[HelpOptions]
; WriteHelpProjectFile = Yes (write each time) or No; if no setting,
; write only if the file does not already exist.
WriteHelpProjectFile = Yes
```

The values you can specify for WriteHelpProjectFile have the following effects:

Yes	If the .hbj file is present, Mif2Go overwrites it.
No	Mif2Go does not overwrite the .hbj file.
(none)	If the configuration file contains no WriteHelpProjectFile setting at all, Mif2Go writes an .hbj file, but <i>only if the .hbj file is not already present</i> .

Mif2Go closes the .hbj file after writing it; however, if you had the .hbj file open in Help Workshop when **Mif2Go** rewrote it, you could get an access violation. (Notepad would just rewrite the old file over the rewritten one.)

If you set WriteHelpProjectFile=Yes and then later decide to modify the .hbj file, be sure to set WriteHelpProjectFile=No; otherwise your edits will be wiped out the next time you convert.

If you use Help Workshop to make changes that are not reflected in the configuration file, and they are changes you want to keep, you can prevent **Mif2Go** from overwriting them by setting WriteHelpProjectFile=No.

8.2.7 Accommodating platform differences

You must specify a few [HelpOptions] settings according to the platform on which your WinHelp file will be used:

```
[HelpOptions]
; Altura = No (default) or Yes (Altura QuickHelp for Mac)
Altura=No
; HyperHelp = No (default) or Yes (Bristol HyperHelp for UNIX)
HyperHelp=No
; ForceBmc = No (default) or Yes (use bmc, not bml, for HyperHelp)
ForceBmc=No
; HelpSectionBreaks = Yes (default) for sect break before each topic,
; or No for Altura (filter strips table format from topic titles)
HelpSectionBreaks=Yes
```

*Windows
9x/ME/NT/2000*

The default settings work for all 32-bit Windows platforms:

```
[HelpOptions]
Altura=No
HyperHelp=No
ForceBmc=No
HelpSectionBreaks=Yes
```

Macintosh

If you are targeting the Macintosh platform, and you are using Altura QuickHelp, set Altura=Yes. This setting does not work for regular Windows versions, so expect to run **Mif2Go** twice to produce both forms. However, you might find that Altura QuickHelp *does* work when you set HelpSectionBreaks=No and Altura=No, in which case you can get by with one version instead of two.

UNIX

For UNIX users, **Mif2Go** has a setting for Bristol HyperHelp: HyperHelp=Yes. Unfortunately, HyperHelp has trouble with WMF graphics. You must use a graphics conversion program (see §5.7.2.3 [Using third-party graphics converters](#) on page 130) to convert all WMF graphics to BMP graphics. If you choose to do this, **Mif2Go** can change the file names for the graphics in the .rtf accordingly:

```
[Graphics]
NameWMFsAsBMPs=Yes
```

Also set [HelpOptions] ForceBmc=Yes, to change all bml references to bmc.

If you are using HyperHelp you might need to prevent multiple interword spacing when there are index markers in text. If you see extra space at index markers, try

EndFtnWithSpace=No:

```
[HelpOptions]
; EndFtnWithSpace = Yes (Help default) or No (HyperHelp default)
EndFtnWithSpace=Yes
; FootnoteSpace = After (the } after the symbol, default),
; Before, or None
FootnoteSpace=After
```

8.2.8 Setting basic WinHelp options in the configuration file

If you are setting up a WinHelp conversion by editing the configuration file instead of using the *Set Up WinHelp Project* dialog, you must provide your own WinHelp.hpj file (see §8.2.5 [Understanding initial set-up requirements](#) on page 246), and add or modify the configuration settings as follows:

1. For each heading format that starts a topic, specify properties:

```
[HelpStyles]
; style = key list, where list members are separated
; by spaces only
ParaFmt = Property1 Property2 Property3 ...
```

where:

ParaFmt is the name of the FrameMaker heading format
PropertyN is a help style attribute, such as Topic, Browse, or Key.

2. For each topic-starting heading format that should appear in the .cnt file, specify a Contents level. For example:

```
[HelpCntStyles]
ParaFmt=Contents level
```

See §8.13.2.1 [Understanding WinHelp contents level numbers](#) on page 289.

3. Specify whether you want a combined .cnt file:

```
[HelpOptions]
; MakeCombinedCnt = Yes (default, when processing from open book)
MakeCombinedCnt=Yes
```

4. Specify whether you want to run the help compiler automatically as the last conversion step (*not* recommended for large projects); if so, provide a name for the help project file:

```
[Automation]
; CompileHelp = No (default, run help compiler separately), or Yes
CompileHelp=Yes

[HelpOptions]
; HPJFileName = name of .hpj to use when compiling help
HPJFileName=myproj.hpj
```

The default value of HPJFileName is the name of your FrameMaker book.

To have **Mif2Go** copy the .hpj file to another directory after generating output files, specify the following:

```
[Automation]
WrapAndShip=Yes
; WrapPath = path to dir for compiling and distribution,
; default is output dir
WrapPath=.\help
```


See §35.6 [Assembling files for distribution](#) on page 961.

8.2.9 Including ObjectIDs in WinHelp

You can specify whether **Mif2Go** includes the FrameMaker ObjectIDs from your document in RTF output for WinHelp, to create link targets or for other purposes:

```
[HelpOptions]
; ObjectIDs = Referenced (default, used by TOC or IX), None, or All
ObjectIDs=Referenced
```

ObjectIDs values have the following effects:

Referenced	Mif2Go includes in the output every ObjectID in your document that serves as a target of a link.
None	FrameMaker ObjectIDs are omitted from the output.
All	Every ObjectID in your document is included in the output.

The default setting allows you to have active cross-reference and hypertext links in WinHelp. For additional settings to activate or suppress these links, see:

§8.4 [Converting cross references](#) on page 259

§8.7 [Converting generated files for WinHelp](#) on page 265

§8.9 [Creating jumps and pop-ups for WinHelp](#) on page 272

For information about ObjectIDs in FrameMaker, see:

§5.3 [Identifying files and objects](#) on page 117

8.2.10 Handling page breaks and section breaks

WinHelp uses page and section breaks for a radically different purpose from print RTF. Page breaks are reinterpreted to mean “start of topic”, and a definite sequence of codes must follow the topic start in the prescribed order. **Mif2Go** creates this sequence automatically when you specify `[HelpStyles]ParaFmt=Topic` (see §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268).

Section breaks are used as an undocumented modifier to page breaks; they permit the help compiler to avoid going into convulsions if the first thing after a page break is a table (as is often the case in FrameMaker for fancy heading designs). Unless you are converting to WinHelp on an Altura system, use the default setting for section breaks:

```
[HelpOptions]
; HelpSectionBreaks = Yes (default) for sect break before each topic,
; or No for Altura (filter strips table format from topic titles)
HelpSectionBreaks=Yes
```

The following options have no use in WinHelp; set them as indicated (the WinHelp defaults):

```
[HelpOptions]
PageBreaks=Remove
KeepSectBreaks=No
```

8.2.11 Providing multiple .hlp files

You can provide multiple .hlp files instead of a single .hlp file. You must set configuration options to ensure the links between .hlp files work as expected. See:

§8.4.2 [Specifying cross-reference destination files](#) on page 259

§8.13.2.5 [Referencing multiple help files from contents](#) on page 291

As an alternative, you can provide a main .hlp file with contents entries that link to other .hlp files; see §7.11 [Setting up a dynamic modular Help system](#) on page 241.

For more information, see **Designing your Help system > Planning your Help system > Designing for multiple Help files** in Help Workshop *Help Topics: Help Author's Guide*.

8.2.12 Integrating WinHelp from RoboHelp

If you have RoboHelp installed on your system, you can integrate a WinHelp DLL created with RoboHelp into your **Mif2Go**-produced WinHelp project.

1. Use RoboHelp to generate a WinHelp 2000 project of any size, even one short file.
2. Open the resulting .hbj file in a text editor, and look for the sections that specify the start-up code.
3. Copy the start-up code sections, unaltered, into the .hbj file produced by **Mif2Go**.

8.2.13 Compiling a WinHelp project

To compile the RTF files **Mif2Go** produces, you need Microsoft Help Workshop (hcx.exe). *Make sure you have the latest version of Help Workshop*. If hcx.exe is not on your system PATH, you must tell **Mif2Go** where to find it:

```
[HelpOptions]
; Compiler = path\to\hcx; can include run parameters
Compiler = hcx /c /e
```

You can have the compiler display a copyright statement and a compile date in the WinHelp *Version Information* dialog:

```
[HelpOptions]
HelpCopyright = your copyright statement
HelpCopyDate = Yes
```

For example:

```
[HelpOptions]
HelpCopyright = (c) 2001-2012 Omni Systems, Inc.
HelpCopyDate = Yes
```

These settings resulted in the following, displayed in the WinHelp *Version Information* dialog:

```
(c) 2001-2012 Omni Systems, Inc.
Monday, February 20, 2012 18:24:39
```

When you check **Compile Help** in the **Mif2Go Export** dialog (see §3.6 [Converting documents](#) on page 82), or specify the following options in the configuration file, **Mif2Go** automatically runs the WinHelp compiler after generating output files:

```
[Automation]
; CompileHelp = No (default, run Help compiler separately), or Yes
CompileHelp=Yes
```

To have **Mif2Go** copy the .hbj file to another directory for compiling, specify the following:

```
[Automation]
; WrapPath = path to dir for compiling and distribution,
; default is output dir
WrapPath=.\help
```

See §35.6 [Assembling files for distribution](#) on page 961.

*Compiling from
Mif2Go not
recommended*

Despite the fact that a **Compile Help** option appears in the **Mif2Go Export** dialog (see [Figure 3-5](#) on page 83), it is not a good idea to compile help from within **Mif2Go**. *If your file is large, you will likely encounter a memory allocation failure.*

*Compile from
Help Workshop*

Instead, open Help Workshop. From within Help Workshop open *MyBook.hpj*; to compile your help project, click the button that looks like a food grinder. You can keep Help Workshop open while you are using **Mif2Go**, and click the compile button whenever you make a new set of RTF files. This method is simple to use, avoids memory problems, and shows you helpful diagnostic messages.

If you still get memory errors, or if compilation does not happen, check the version of Help Workshop you are actually using (not just the *hwc.exe* you see in Explorer): double-click *MyBook.hpj* and look at the number displayed when you click **Help > Version...**; it should be 4.03.0002.

*Help compiler
must be on the
system PATH*

That said, if you really must compile from within **Mif2Go**, make sure the Windows system can find file *hcrtf.exe*. This is best accomplished by including the name of the directory where you installed Help Workshop in the system PATH by modifying environment variable PATH. Or, you can copy the *.exe* files (at least *hwc.exe* and *hcrtf.exe*) and the *.dll* to the *\windows\system* directory, or to some other directory that is already on the system PATH.

Also, you might want to specify the following setting, so you can see any error messages that result:

```
[Automation]
; KeepCompileWindow = No (default)
; or Yes (so any error messages can be seen)
KeepCompileWindow=Yes
```

When *KeepCompileWindow=Yes*, a system window opens when the compiler runs. If there are no compilation errors, you will see only a command prompt when compilation finishes. You must dismiss the window before **Mif2Go** can continue processing.

*Tell the Help
compiler where to
find graphics*

If your graphics are in more than one place, you can add multiple *BMROOT=* entries to the [OPTIONS] section of your Help Project (*.hpj*) file. For example:

```
[OPTIONS]
BMROOT=..\MyGraphics
BMROOT=..\Test\Graphics
```

*Edit the .hpj if you
add / remove /
rename files*

If you add, remove, or rename any files in your FrameMaker book after **Mif2Go** has generated the *.hpj* file for that book, before you compile WinHelp you must edit the [Files] section of the *.hpj* file to reflect those changes. Use a text editor such as Notepad.

8.2.14 Checking WinHelp RTF files for Mif2Go version

If you recently installed a **Mif2Go** upgrade or beta version, after you run **Mif2Go**, check to make sure the latest version was actually used to produce RTF output. Windows sometimes caches DLLs, and does not always use a newly replaced DLL until after the system is rebooted.

Open an RTF output file in Word and choose **File > Properties > Comments**. You should see a line like the following:

```
DCL filter dwrtf, Ver 3.3 m194b r278b
```

The last two entries identify the build numbers of the **Mif2Go** *drmf.dll* and *dwrtf.dll* components that were used to create the RTF file. See §D.2.9 [Check your version of Mif2Go](#) on page 1034.

8.3 Converting text

In this section:

- §8.3.1 [Converting formats for WinHelp](#) on page 252
- §8.3.2 [Converting special characters](#) on page 254
- §8.3.3 [Removing unused formats and fonts](#) on page 257
- §8.3.4 [Converting autonumbers](#) on page 257
- §8.3.5 [Replacing paragraph or character content](#) on page 257
- §8.3.6 [Specifying text color](#) on page 258
- §8.3.7 [Converting footnotes](#) on page 258

8.3.1 Converting formats for WinHelp

Most format settings for print RTF apply also to WinHelp RTF; these settings are described in §6 [Converting to print RTF](#) on page 141, specifically in §6.7 [Converting paragraph and character formats](#) on page 158. The settings described here include those that are exclusive to WinHelp, or that differ in usage from the same-named settings for print RTF.

Caveats Observe the following restrictions on format names in your FrameMaker document. *Do not use:*

- **The same name for two different types of format;** for example, a paragraph format named *Body* and a character format also named *Body*.
- **Two formats that differ only in case or in spacing;** for example, *Heading 1*, *Heading1*, *heading 1*, and *heading1* are identical in RTF usage.

In this section:

- §8.3.1.1 [Converting paragraph formats](#) on page 252
- §8.3.1.2 [Suppressing unwanted paragraphs](#) on page 253
- §8.3.1.3 [Converting character formats](#) on page 254

See also:

- §2.2 [Naming FrameMaker formats](#) on page 66

8.3.1.1 Converting paragraph formats

Many of the settings described in §6.7 [Converting paragraph and character formats](#) on page 158 for print RTF apply equally to WinHelp output.

Sidehead formats In [HelpOptions], set Sideheads=Left. This is usually the best choice for WinHelp:

```
[HelpOptions]
; Sideheads = Left (default), Indent (looks more like doc), or Normal
Sideheads=Left
```

Run-in headings By default, for WinHelp output **Mif2Go** separates each run-in heading from its following paragraph. To have **Mif2Go** emulate the FrameMaker run-in heading format in WinHelp:

```
[HelpOptions]
; RunInHeads = Runin (Word default) or Normal (help default)
RunInHeads=Runin
```

When RunInHeads=Normal, for WinHelp **Mif2Go** inserts a carriage return between the run-in heading and the paragraph that follows.

When RunInHeads=Runin, the following paragraph starts on the same line as the run-in heading, as it does in FrameMaker. However, this setting does not work correctly if you

use the run-in heading for a WinHelp topic start, because topic-start formats disappear from the body of the topic.

Reference frames You can choose whether or not to include reference frames defined for paragraph formats, and if so, whether to use the actual reference-page graphic or just its name. You might want the name if it is descriptive, such as “Note” or “Caution”.

```
[HelpOptions]
; RefFrames =
;   Graphic (show FrameAbove and Below),
;   Text (name only), or
;   None
RefFrames=Graphic
```

If some of your paragraph formats include a line above or below, and you do not want those lines to appear in your help file:

```
[HelpOptions]
RefFrames=None
```

If you specify Text to include just the name, also specify a format (FrameMaker paragraph format) for the name. For example:

```
[HelpOptions]
; RefFrameDefFormat = the format to be used for Text reference frames
RefFrameDefFormat=AlertHead
```

Tabs **Mif2Go** uses font metrics (see §6.9.3 [Specifying font types](#) on page 167) to convert tabs. You can specify a default tab width in twips (twentieths of a point):

```
[HelpOptions]
; DefTabWidth = 0 (default, ignore undefined tabs)
;   or twips (720 for 0.5")
DefTabWidth=0
```

When too many tabs are defined, sometimes the Help Compiler fails with a heap corruption error. **Mif2Go** provides an empirical default maximum of 32 tabs defined per paragraph format:

```
[HelpOptions]
; HelpTabLimit = maximum tab definitions allowed
;   in paragraph formatting
HelpTabLimit=32
```

If you need to increase the limit, do so gradually, and watch out for compiler errors. We have found that 45 fails consistently.

Note: The maximum applies to number of tabs *defined*, even if they are never used.

8.3.1.2 Suppressing unwanted paragraphs

Suppose your FrameMaker document contains manually inserted page-oriented navigation aids, such as the text “(Continued)” when a procedure breaks across a FrameMaker page boundary. To prevent this text from appearing in RTF output, you can use conditional text, or you can do the following:

1. Use a special paragraph format (for example, *Continuation*) for all instances of the text in your document.
2. In the configuration file, assign property Delete to the paragraph format:

```
[HelpStyles]
; Delete is used to remove displayable text
Continuation=Delete
```

Note: If a table or graphic is anchored in a paragraph whose format is assigned the Delete property, the table or graphic is retained, and only the text of the paragraph is deleted.

8.3.1.3 Converting character formats

Mif2Go can output FrameMaker character formats as Word character styles, instead of as overrides. Use this feature only if you anticipate needing to revise the RTF text:

```
[HelpOptions]
; CharStylesUsedInText = No (help default, no cs codes allowed)
CharStylesUsedInText = Yes
```

Most character-format settings for print RTF apply also to WinHelp RTF; these settings are described in §6.7.8 [Converting character formats](#) on page 163. The settings described here are those that are exclusive to WinHelp, or that differ in usage from the same-named settings for print RTF:

[Small Caps](#)

[Underlining used as an override](#)

Small Caps Text in Small Caps style might be too small to read easily, reducing the size of the original caps. If this happens, you can turn off the Small Caps style:

```
[HelpOptions]
; SmallCaps = Standard (default), or None
SmallCaps=None
```

**Underlining used
as an override**

By default, **Mif2Go** eliminates underlining that was applied to text in FrameMaker via format overrides; that is, underlining is removed unless it is defined to be part of a character or paragraph format:

```
[HelpOptions]
; AllowLiningOverrides = No (default,
; underline change only in formats)
AllowLiningOverrides=No
```

This prevents confusion of random underlined text with links, which are underlined automatically in WinHelp. When someone clicks underlined text and nothing happens, it looks as though a link is broken. You can change this setting to **Yes** to keep the underlining.

8.3.2 Converting special characters

Not all characters in FrameMaker text translate as you might expect. For example, WinHelp has very limited capabilities for displaying symbols as part of the text, and typically uses an embedded bitmap containing the image of the symbol.

In this section:

- §8.3.2.1 [Specifying font encoding for non-Western languages](#) on page 255
- §8.3.2.2 [Using one character to represent all characters in a font](#) on page 255
- §8.3.2.3 [Embedding bitmaps in a font](#) on page 255
- §8.3.2.4 [Converting bullets](#) on page 256
- §8.3.2.5 [Converting smart quotes](#) on page 256
- §8.3.2.6 [Converting hard hyphens](#) on page 257
- §8.3.2.7 [Converting high ASCII characters](#) on page 257

8.3.2.1 Specifying font encoding for non-Western languages

For WinHelp (and for Word), the default font encoding for font types 5 and 6 (see [Table 6-4](#) on page 168) is `fcharset2`; for all other font types, the default encoding is `fcharset0`. You can specify a different font encoding for text in which the “high ASCII” character set contains characters other than the European accented characters (represented by `fcharset0`). For example, to specify Cyrillic font encoding:

```
[FontEncoding]
; Font name = value to use in font table for fcharset
Times New Roman CYR=204
```

This setting tells WinHelp to display Cyrillic characters in place of accented characters for the high ASCII code points.

See also:

§6.9.3 [Specifying font types](#) on page 167

§6.9.4 [Specifying font encoding for non-Western characters](#) on page 168

8.3.2.2 Using one character to represent all characters in a font

In WinHelp 4, you can specify a normal font character to be used in place of any character in the font; this method is typically used to set a bullet. You do this by changing the `[FontTypes]` value for the font to the decimal value of the character you want to use as the replacement, such as `Wingdings=40` to use “*” in place of any Wingding character:

```
[FontTypes]
Wingdings=40
```

See §6.9.3 [Specifying font types](#) on page 167 for more information about font types.

8.3.2.3 Embedding bitmaps in a font

You can specify a font for which you want certain characters changed to in-line bitmaps in the WinHelp file. You embed bitmaps in the font as follows:

1. Choose a font to represent bitmaps.
2. Assign a different character in that font to each bitmap.
3. Place the assigned characters—in that font—in your FrameMaker document, wherever you want the bitmaps to appear in WinHelp.

In the configuration file you identify the font (`BitmapFont=fontname`), then map the characters. You can represent a character either as itself or as its decimal numeric value. For example:

```
[BitmapChars]
; BitmapFont = name of font to check for chars to map here
BitmapFont=Algerian
; following chars are remapped when in BitmapFont in WinHelp only
; sample mappings are all to bitmaps supplied in Help Workshop
; * = bullet, using the decimal numeric value of the character
40=bullet.bmp
; A = arrow, using the character itself (printable, not ";" or "[")
A=prcarrow.bmp
; C = closed book
C=closed.bmp
; D = document
68=document.bmp
; O = open book
79=open.bmp
```

```
; S = step
S=onestep.bmp
```

Add a line for each character you want to map, such as:

```
40=bullet.bmp
```

to make all asterisks in that font turn into in-line calls for a bullet image. Any characters you do *not* map remain themselves. To use the semicolon “;” or the left bracket “[”, you must precede the character with a backslash to avoid conflict with configuration-file conventions.

Make sure you tag only one character at a time with the bitmap font, and watch out for preceding or following spaces in the same font, which would disable the bitmap usage.

When you use actual letters as [BitmapChars] keys, they are case insensitive; “A” is mapped the same as “a”. Using the decimal numeric value of a character as the key instead has two advantages:

- You can map high-bit-set characters, which are often used for symbols.
- You can assign different bitmaps to uppercase and lowercase characters.

8.3.2.4 Converting bullets

Bullets can be put in as characters in WinHelp 4, according to the documentation. In some cases, bullet characters work better than bitmaps, which can interfere with paragraph indents and spacing:

```
[HelpOptions]
; Bullets = Help (default, bmc reference) or Standard (using '95)
Bullets=Help
; BMPsForDingbats = No (default)
; or Yes (to use bullet.bmp for deco fonts)
BMPsForDingbats=No
```

The default is Bullets=Help, which uses in-line embedded bitmaps. To use characters instead, set Bullets=Standard. Dingbats and Wingdings usually work in WinHelp 4, but sometimes they come out as text characters. If that happens, use the bitmap instead, by setting BMPsForDingbats=Yes.

If you want to use a different name for your bullet bitmap, you can change the name here:

```
[Graphics]
BulletFile=mybull.bmp
```

8.3.2.5 Converting smart quotes

The WinHelp 4 documentation says smart quotes (“curly” quotes) work just fine. They do not. They often cause a spurious error message about “unmatched braces” in RTF output. **Mif2Go** offers three options for dealing with this problem:

```
[HelpOptions]
; Quotes = Help (default), Standard (curly, may cause errors),
; or Numeric
Quotes=Help
```

These settings have the following effects:

Help	Changes curly quotes to straight quotes, which do work in WinHelp 4.
Standard	Leaves them as curly quotes; sometimes works in very small .hlp files.

Numeric	Uses “” syntax for the quotes, and might work for curly quotes where Standard does not.
---------	---

8.3.2.6 Converting hard hyphens

Because WinHelp 4 does not consistently recognize hard hyphens, **Mif2Go** automatically changes them to regular hyphens.

8.3.2.7 Converting high ASCII characters

If you use em dashes and many other high ASCII characters, **Mif2Go** must replace them with in-line embedded bitmaps containing the image of the symbol. If you are using Help Workshop for Win9x/ME/NT/2000-compatible .hlp files, several bitmaps are built in to the Help Compiler; see the Help Author's Guide (the “Help” in Help Workshop) for details.

If you are using HC.EXE for Win3.1 compatibility, you are on your own. **Mif2Go** provides `bullet.bmp`; you will have to acquire or create any other bitmaps.

8.3.3 Removing unused formats and fonts

Mif2Go keeps track of which paragraph and character formats and which fonts are actually used in the output file, and can remove unused formats and fonts. This is the default for WinHelp, because WinHelp RTF files are rarely edited again before they are compiled:

```
[HelpOptions]
; RemoveUnusedStyles = Yes (default for WinHelp) or No
RemoveUnusedStyles=Yes
; RemoveUnusedFonts = Yes (default for WinHelp) or No
RemoveUnusedFonts=Yes
```

8.3.4 Converting autonumbers

By default, **Mif2Go** includes FrameMaker paragraph autonumbers as text in WinHelp output. To omit FrameMaker autonumbers:

```
[HelpOptions]
; WriteAnums = Yes (default) or No (omit FrameMaker autonumbers)
WriteAnums = No
```

When `WriteAnums=Yes` (the default), **Mif2Go** converts FrameMaker autonumbers to text and includes them in WinHelp output.

When `WriteAnums=No`, FrameMaker autonumbers are omitted.

8.3.5 Replacing paragraph or character content

You can direct **Mif2Go** to replace the content of a paragraph, or of a character-formatted span of text, with arbitrary RTF code. For example, to replace page numbers with graphics in a generated file to be included in WinHelp output:

```
[HelpStyles]
; Replace deletes, and also puts out the RTF in [HelpReplacements]
IOMpgnum=Replace
```

You specify the replacement RTF code as a property of the format in question, in the following section. For example:

```
[HelpReplacements]
; Replace causes the insertion of the corresponding raw RTF code below
;   in place of the original content of the named para or char format
IOMpgnum=\{bmc document.bmp\}
```

This feature is used in the WinHelp version of the **Mif2Go User's Guide**, to replace page numbers in keyword indexes with bitmap graphics; see §8.7.2 [Converting indexes and lists of marker references](#) on page 266.

8.3.6 Specifying text color

Mif2Go converts FrameMaker color definitions from CMYK to RGB, and adds them to the RTF color table so that they look the same in WinHelp. You can choose whether to retain text colors:

```
[HelpOptions]
; TextColor = 0 all black (help default) or 1 (as is)
TextColor=0
```

If you use the colors only for ease of editing in FrameMaker, and want them to come out black in the converted files, set `TextColor=0`. This is the default for WinHelp conversions, because color is used in a consistent way in WinHelp to indicate hotspots, and other uses might confuse the reader. However, you can override the default with `TextColor=1`, so that your text colors appear in the WinHelp file.

Note: The `TextColor` setting does not affect the use of color in graphics, including graphics text (such as callouts). The RTF color table also does not affect the colors used in graphics; the color information is embedded in the graphic metafile.

8.3.7 Converting footnotes

Mif2Go can convert a FrameMaker footnote to a WinHelp jump (the default) or to a WinHelp pop-up, instead of leaving it as a footnote:

- Jump:* The footnote number is a hotspot; click it, and you jump to the footnote, which is placed at the bottom of the topic. You can also view the footnote by scrolling to the bottom of the page; if the topic is short, you might not need to scroll.
- Pop-up:* The footnote appears in a pop-up window by itself when you click the footnote number; it does not appear on the topic page at all. This might be desirable in long tables, to give added bits of information for selected items without scrolling.

The default is to convert footnotes to jumps. You can also specify that footnotes should remain as is, or appear embedded in the text between brackets in place of the footnote number:

```
[HelpOptions]
; Footnotes = Standard, Embed (between []), Jump, Popup, or None
; default is Jump, which looks more normal than Popup
Footnotes=Jump
```

To separate footnotes from text at the end of the topic when `Footnotes=Standard`:

```
[HelpOptions]
; FootnoteSeparator = RTF to use for separator above footnotes at the
; bottom of the page, can be a macro reference, default none
FootnoteSeparator=
\n\pard\plain\fs20\emdash\emdash\emdash\emdash\par\n
```

This setting must be all on one line, even though it might not appear that way here.

8.4 Converting cross references

By default, **Mif2Go** converts cross references into jumps, including interfile jumps where needed.

In this section:

§8.4.1 [Creating help context markers](#) on page 259

§8.4.2 [Specifying cross-reference destination files](#) on page 259

§8.4.3 [Specifying cross-reference jump destinations](#) on page 260

§8.4.4 [Specifying WinHelp options for cross-reference formats](#) on page 260

§8.4.5 [Limiting cross-reference text](#) on page 261

8.4.1 Creating help context markers

Mif2Go can convert cross references to help context markers; this is the default for WinHelp:

```
[HelpOptions]
; Xrefs = Help (make context markers) or None (plain text)
Xrefs=Help
```

You can specify whether to use the FrameMaker numeric cross-reference ID (the default), or the full text of the cross reference:

```
[HelpOptions]
; XrefType = = Numeric (default) or Full (use only to eliminate dupes)
XrefType=Numeric
```

If you specify `XrefType=Full`, you get the complete referenced text in the WinHelp context marker. Such text can easily exceed the 63-character size limit for cross-reference IDs in WinHelp, and might contain characters that are not valid in WinHelp IDs.

If you specify `XrefType=Numeric` (or if you omit the setting entirely) **Mif2Go** uses FrameMaker ObjectIDs instead of the full cross-reference text.

Duplicate references

In the great majority of cases, `XrefType=Numeric` produces the correct result. However, you run a very small chance of encountering duplicate cross-reference ID numbers, which cause the WinHelp compiler to complain. If this happens, you must replace one of each pair of duplicate ID numbers in your FrameMaker document. To find the duplicates in FrameMaker, generate an Index of Markers (IOM) for your document, selecting only cross-reference markers, and checking **Create Hypertext Links**. A duplicated number has two references listed instead of one.

8.4.2 Specifying cross-reference destination files

If all your cross references are to sources within the same `.hlp` file, **Mif2Go** can process them without further information from you. If you have cross references to other files, you must specify into which `.hlp` files the original source files will be placed.

Single file

When all source files are going into the same `.hlp` file, you can use a one-step setting:

```
[HelpOptions]
XrefFileDefault=helpfilename
```

Multiple files

When you create multiple interlinked `.hlp` files, you must specify a mapping for each external file name that is specified in the helpset (but not in the current `.hlp` file) to the name of the `.hlp` file that contains the corresponding topic. You must insert this information in the configuration file for each helpset in the interlinked group of helpsets. Do not include paths or file extensions. For example:

```
[HelpXrefFiles]
; file name in xref = file name for .hlp
intro=help1
chap1=help1
chap2=help2
```

- Default file** **Mif2Go** checks [HelpXrefFiles] for each cross reference to another file; if the file name is not listed, **Mif2Go** retains the original file name. Therefore, it is important to list the names of *all* source files included in each .hlp file.
- Links to file names** What this does is cause the .hlp file name to be added to any links that reference the other files named. You do not need the mappings for the files going into the .hlp file you are constructing, because those do not require the extra information. You do need the mapping for files outside the .hlp file you are currently constructing. However, having the entire set present for all files is harmless.
- Links from contents** If you are merging multiple help files into a set, also consider contents entries; see §8.13.2.5 [Referencing multiple help files from contents](#) on page 291.

8.4.3 Specifying cross-reference jump destinations

Mif2Go uses a reference number to produce a jump from a cross reference. For interfile cross references to or from the help file you are producing, you must specify the names of all the topic files involved, in the form *topicfile=helpfile*; for example:

```
[HelpXrefFiles]
; file name in xref = file name for .hlp
chap1=chaphelp
```

You can specify a file extension for the destination help file; the default is hlp:

```
[Setup]
; FileSuffix = suffix to use (no leading dot)
; when converting [HelpXrefFiles] xrefs
FileSuffix=hlp
```

You can specify a name (without file extension) to use for any topic files not listed, as *XrefFileDefault=helpfilename*. You can use this setting for your own usual help-file name, so you do not have to name this file explicitly under [HelpXrefFiles]. For example:

```
[HelpOptions]
; XrefFileDefault = name of file to use for missing XrefFiles
XrefFileDefault = ugmif2go
```

8.4.4 Specifying WinHelp options for cross-reference formats

You might not want every cross reference in your document to become a link in WinHelp. You can choose to have **Mif2Go** delete cross references of a certain format, or convert them to text. For example:

```
[XrefStyles]
; xref format name = properties (Delete or Text)
; if omitted treated as link
Heading & Page=Text
Page=Delete
```

In this example, **Mif2Go** would render any cross reference that uses the *Heading & Page* format as plain text rather than as a link. **Mif2Go** would also delete any cross reference that uses the *Page* format.

8.4.5 Limiting cross-reference text

FrameMaker cross references can be quite long. If you are using the full text, not just the number part, referencing them can be a problem in WinHelp. You can specify a limit to the length of cross-reference text, and **Mif2Go** will truncate any that are longer. Set the limit shorter to save space, or longer to eliminate duplication when text is truncated:

```
[HelpOptions]
; XrefLenLimit = 64 (default max length
; for xref identifiers, truncate)
XrefLenLimit = 64
```

See also §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274.

8.5 Converting tables to WinHelp RTF

You can adjust the way tables appear in WinHelp, to a limited extent; the RTF way of making tables is primitive, and WinHelp takes away most of the few options. You can also disassemble tables and convert the rows to topics.

In this section:

§8.5.1 [Positioning tables and table titles](#) on page 261

§8.5.2 [Adjusting table appearance](#) on page 261

§8.5.3 [Converting table rows to topics and table cells to pop-ups](#) on page 262

8.5.1 Positioning tables and table titles

To adjust how tables are positioned in relation to surrounding text in WinHelp:

```
[Tables]
; ShiftWideTablesLeft=Yes (default, unindent overwidth tables) or No
ShiftWideTablesLeft=Yes
; TableWidthsFixed=Yes (default) or No (centered tables are variable)
TableWidthsFixed=Yes
```

Set `TableWidthsFixed=No` to cause tables that are centered in FrameMaker to be adaptively sized to the window width in WinHelp. All other converted tables are left-aligned; WinHelp does not support right-aligned tables.

Note: Setting `TableWidthsFixed=No` makes only *centered* tables adaptive in size. This is a WinHelp rule; no other table alignment results in adaptive sizing.

Titles To position table titles:

```
[Tables]
; TableTitles = 0 to leave alone, 1 to put at top, 2 to put at bottom
; put at top when used as topic titles or jump targets
TableTitles=1
```

Usually the best position for WinHelp is above the table.

8.5.2 Adjusting table appearance

You can fine-tune a few aspects of table appearance in WinHelp:

[Rules, fill, line breaks](#)

[Graphics](#)

[Column width](#)

[Column straddles](#)

Rules, fill, line breaks

These settings apply to all tables in the document:

```
[Tables]
; TableRules = None (help default), or one of the Box types:
; Box, Double, Thick, Shadow, Para, or Variable
TableRules=None
; TableFill = AsIs (default, shading is unavailable), ColorOnly, None
TableFill=AsIs
; ForceTableLineBreaks = No (default) or Yes (make soft breaks hard)
ForceTableLineBreaks=No
```

When ForceTableLineBreaks=Yes, **Mif2Go** turns line wraps in table cells into line breaks in WinHelp, which often does a poor job of wrapping text in table cells.

Graphics

If you have frames anchored inside a table cell that do not appear in the output, specify the following:

```
[Tables]
; TableGraphics = Standard (default, in cell), None, or Outside
; applies only to non-inline and non-runin frames anchored in cell
TableGraphics=Outside
```

Column width

Sometimes the font used in WinHelp is larger than the original font used in FrameMaker. The size increase can make text too large for a table cell, causing it to run into the next table cell. **Mif2Go** has two settings available to fine tune the cell size when this happens. You can adjust the table column width as a percentage of the original width, or change the width by a number of twips (twentieths of a point):

```
[Tables]
; TblColWid* rescales all table column widths in the file, using:
; Pct as the percentage to apply to the original size, 0-32766
; Add as the twips to add to the scaled result; neg to subtract
TblColWidPct=100
TblColWidAdd=0
```

You can use both in combination; for example, to add 20% + 360 twips:

```
[Tables]
TblColWidPct=20
TblColWidAdd=360
```

To specify the full width of tables for which column widths are percentages:

```
[Tables]
; TblFullWidth is the size in twips used to compute column widths when
; widths are given as percentages, default of 9360 twips (6.5 inches).
TblFullWidth=9360
```

Column straddles

By default, **Mif2Go** combines table cells that straddle columns into a single cell in WinHelp; however, you can override the default:

```
[Tables]
; MergeStradCells = Yes (default, combine col-straddling cells) or No
MergeStradCells=No
```

Note: For Word output, the default value of MergeStradCells is No (the opposite of the default for WinHelp); see §6.13 [Converting tables to print RTF](#) on page 184.

8.5.3 Converting table rows to topics and table cells to pop-ups

Mif2Go can remove the formatting from a table and make each row into a topic:

```
[Tables]
; StripTables = No (default) or Yes (when every row is a new topic)
StripTables=Yes
```

This setting applies to all tables in a document (or in a single file, if you convert that file separately); use it only to disassemble *all* the tables. The text in the first column of each table row becomes the topic title if you designate the paragraph format used in the first column as Topic:

```
[HelpStyles]
First_column_format=Topic
```

Before stripping a table, **Mif2Go** adds an RTF paragraph break (`\par`) to each cell to mark the end of the cell content after the table formatting is gone. If you are using the cell content for a pop-up (see §8.8.1 [Creating WinHelp topics](#) on page 267), the presence of the `\par` causes extra space around the pop-up text. To reduce the space, you can set the following option:

```
[Tables]
; StrippedCellPar = Yes (default, add \par after cells)
; or No (omit it)
StrippedCellPar=No
```

8.6 Managing graphics for WinHelp

WinHelp understands only two graphics formats: WMF and BMP. Graphics in other formats must be converted. Although it may seem that other formats such as GIF work for WinHelp, actually any format other than WMF or BMP is converted by the WinHelp compiler, using a Microsoft Office filter; not the best process.

If your document contains embedded graphics (copied into FrameMaker instead of imported by reference), **Mif2Go** can export the embedded graphics in their original format; see §31.2.3 [Exporting and converting embedded graphics](#) on page 877.

To exclude graphics entirely, or display only graphics file names, see §31.3.2.5 [Excluding graphics from RTF output](#) on page 895.

In this section:

- §8.6.1 [Choosing a graphics format for WinHelp](#) on page 263
- §8.6.2 [Avoiding the GDI resource leak](#) on page 264
- §8.6.3 [Positioning graphics in WinHelp](#) on page 264
- §8.6.4 [Displaying graphics in pop-ups for WinHelp](#) on page 265

See also:

- §5.7 [Processing graphics](#) on page 126
- §8.9.3.4 [Embedding hotspots in graphics for WinHelp](#) on page 275
- §31 [Working with graphics](#) on page 869

8.6.1 Choosing a graphics format for WinHelp

The format WinHelp likes best is WMF, which can contain both vector and bitmap elements. For screenshots, use the same resolution at which they were taken, 96DPI; more or less will result in unreadable text.

*WMF good
(on Win NT/2000)*

If your graphics are in a vector format, the WMF equivalents that **Mif2Go** produces by default as part of the conversion render better than BMP graphics. For example, using WMF instead of BMP can make the difference between readable and not for text on a flow chart. And WMF gives the best display, by far, for graphics to which you have added callouts.

*WMF bad
(on Win 9x/ME)*

However, WMF graphics can cause serious problems for users on Windows 9x or Windows ME systems; see §8.6.2 [Avoiding the GDI resource leak](#) on page 264.

If your graphics are in a bitmap format (other than BMP) and you convert them outside of **Mif2Go**, convert to BMP and let **Mif2Go** wrap the images in WMF code; see §5.7.2.3 [Using third-party graphics converters](#) on page 130. Use 256-color BMPs, no higher (no 16-bit or 24-bit BMP images).

*BMP good
(on UNIX or Mac)*

For non-Windows WinHelp, created via Altura on Macintosh or Bristol HyperHelp on UNIX, only BMP is supported; see §8.2.7 [Accommodating platform differences](#) on page 247.

8.6.2 Avoiding the GDI resource leak

If the WinHelp you create is used on Windows 9x or Windows ME systems, WMF graphics can eat up all GDI resources with amazing speed. An operating-system defect causes a GDI resource leak, which can crash the system when a user pages through a WinHelp file that contains WMF graphics.

The GDI resource leak happens with all WMF graphics, regardless of source. This can rule out WMF as a graphics format if you have clients using Windows 9x or Windows ME. The problem does not occur on Windows NT, Windows 2000, or Windows XP; nor does it occur with formats other than WMF.

If you have FrameMaker graphics that include just a single bitmap and no other elements, you can get around the problem by mapping any non-BMP formats to BMP (see §31.3.2 [Changing graphics files for RTF output](#) on page 890). Do not direct **Mif2Go** to wrap BMP graphics in WMF (see §31.2.6 [Embedding bitmap graphics in WMF for WinHelp](#) on page 886); that is, make sure `[Graphics]EmbedBMPsInWMFs=No`.

The issue becomes acute when your graphics include callouts or multiple BMP elements. In that case, you have two choices:

- **Use Mif2Go graphics processing.** This is a three-step workaround:
 1. In a first conversion pass, allow **Mif2Go** to create WMF files from the graphics in your FrameMaker document; see §5.7.2.1 [Using Mif2Go native graphics processing](#) on page 128.
 2. Create matching BMP files with an external program that can import the WMF files; see §5.7.2.3 [Using third-party graphics converters](#) on page 130.
 3. In a second conversion pass, set `[Graphics]NameWMFsAsBMPs=Yes` to make the RTF output refer to the BMP files instead of to the WMF files; see §31.3.2.1 [Substituting graphics files for RTF](#) on page 890.
- **Use FrameMaker export filters.** This is a simpler workaround, but the results might be of poor quality:
 - Direct **Mif2Go** to use FrameMaker export filters to make BMP graphics instead of WMF graphics; see §5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129.

8.6.3 Positioning graphics in WinHelp

You might want to override the default positioning of all regular anchored frames (such as screen shots), so you can make them more consistent; for example:

```
[HelpOptions]
; FrameStyle = para style for non-in-line anchored frames
; default is not to specify, which uses the previous para style
FrameStyle=Picture
```

The default is not to specify a format, which causes graphics to use the previous paragraph format. WinHelp users can easily resize the window, so graphics and their captions should stay consistent when that happens.

8.6.4 Displaying graphics in pop-ups for WinHelp

Suppose you want one or more of the illustrations in your document to appear only in pop-ups, instead of in line with the text; and suppose you want the illustrations to appear only when a user clicks the figure caption. **Mif2Go** provides two ways to accomplish this:

- Insert hypertext markers in the anchor and caption paragraphs.
- Assign additional format properties to the anchor and caption paragraphs.

The results of either method are as follows:

- The caption becomes a hotspot.
- The illustration appears in a pop-up window, and is displayed only when the caption is clicked.

Note: For either method to work, graphics to appear in pop-ups must have captions positioned *below* the graphics.

*Using hypertext
markers*

To use hypertext markers for pop-up graphics (see §8.9.5 [Using hypertext links for jumps and pop-ups](#) on page 276):

1. Embed a hypertext **newlink** marker with a unique name in the paragraph that holds the anchor for the graphic.
2. Embed a corresponding hypertext **gotolink** marker in the caption paragraph for the graphic.
3. In the configuration file, set the following:

```
[HelpStyles]
FigAnchor=Topic Slide Scroll NoXScroll
FigCaption=PopOver Green Resume
```

For this to work, every graphic must have a caption paragraph *after* the graphic, or at least some distinct paragraph format immediately following the graphic that can be assigned property *Resume*. Otherwise, the rest of the topic also appears in the pop-up.

*Using format
properties*

To use format properties for pop-up graphics (see §8.9.11.6 [Using unique references for jumps and pop-ups](#) on page 282):

1. Assign property *MakeRef* to the paragraph format that anchors the graphic.
2. Assign property *PrevRef* to the caption paragraph, which must follow the graphic.

Configuration settings are as follows:

```
[HelpStyles]
FigAnchor=Topic Slide Scroll NoXScroll MakeRef
FigCaption=PopOver Green Resume PrevRef
```

8.7 Converting generated files for WinHelp

Mif2Go generates standard WinHelp contents and index entries from your FrameMaker TOC and index markers. However, you might want to include additional generated files in your WinHelp output, such as lists of tables and figures (or other special-purpose paragraph lists), and lists of markers (or other special-purpose indexes).

In this section:

§8.7.1 [Converting lists of paragraph references](#) on page 266

§8.7.2 [Converting indexes and lists of marker references](#) on page 266

8.7.1 Converting lists of paragraph references

For FrameMaker-generated lists of references to paragraphs (such as LOF and LOT), you must make sure the ObjectIDs of all the referenced paragraphs are preserved in WinHelp output (see §8.2.9 [Including ObjectIDs in WinHelp](#) on page 249), with this setting:

```
[HelpOptions]
ObjectIDs=All
```

To make the list entries themselves into links, assign the [HelpStyles]ParaLink property to the list-entry paragraph format; see §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274.

To suppress page numbers, on the FrameMaker Reference page for the generated file you can apply a character format to <\$pagenum> and to all its leading tabs, then assign that character format the [HelpStyles>Delete property.

For example:

```
[HelpStyles]
FigLOF=ParaLink
LOFpgnum>Delete
```

In the WinHelp version of the **Mif2Go User's Guide**, these features are used to make the Figures and Tables entries into links.

8.7.2 Converting indexes and lists of marker references

A list of references to markers can include links to multiple markers for each entry, so you cannot use the [HelpStyles]ParaLink property (see §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274) to make the text of the entry the link. Instead you can either use the original page numbers, or substitute a symbol or graphic.

To substitute something else for page numbers, you must apply a character format to <\$pagenum> on the FrameMaker Reference page for the marker list, and assign to that format the [HelpStyles]Replace property; see §8.3.5 [Replacing paragraph or character content](#) on page 257.

For example:

```
[HelpStyles]
IOMpgnum=Replace
```

You specify RTF code for whatever you want instead of a page number. For example:

```
[HelpReplacements]
IOMpgnum=\{bmc document.bmp\}
```

In the WinHelp version of the **Mif2Go User's Guide** this setting is used to replace the page numbers in indexes with small page icons taken from the built-in bitmap set in Help Workshop. For other options, in Help Workshop choose **Help > Help Topics > Find**, type in *bull*, and go to the first topic offered: "Bitmaps supplied by Help Workshop".

For lists of references to markers, the extra ObjectIDs you must supply for lists of paragraphs are not needed; and in fact, they interfere with the marker links. If you are converting both paragraph lists and marker lists, you must prepare a special file-specific configuration file for each of the marker lists, with this setting:

```
[HelpOptions]
ObjectIDs=Referenced
```


If your document includes more than one marker list, the Help compiler might warn you about duplicate links. This is because the **newlinks** generated automatically by FrameMaker can be identical for multiple marker lists. To differentiate links, you must specify a prefix to use for each marker list. These settings must be placed in a special file-specific configuration file for each marker list.

For example:

```
[HelpOptions]
;IXnewlinkPrefix = prefix to use on newlinks, only in specific .inis
; to differentiate autogenerated newlinks in different Index files
IXnewlinkPrefix=IKH_
```

See §33.1 [Using a different configuration for selected files](#) on page 919 for information about creating individual configuration files.

8.8 Configuring WinHelp topics

To produce the organization and navigation features of WinHelp, you assign topic and hotspot properties to FrameMaker paragraph, character, and cross-reference formats. The first property assigned to each format identifies the role that all text in that format will play in the Help system.

Note: If you are converting via FrameMaker, **Mif2Go** automatically makes a first-pass determination of the most likely assignments; see §1.5 [How Mif2Go works](#) on page 62. Be sure to check these assignments for reasonableness.

In this section:

§8.8.1 [Creating WinHelp topics](#) on page 267

§8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268

§8.8.3 [Configuring topic titles for WinHelp](#) on page 271

8.8.1 Creating WinHelp topics

To create topics, in the configuration file assign property `Topic` to the FrameMaker paragraph format of the heading (or other paragraph) that starts material to be included in a topic:

```
[HelpStyles]
; style = key list, where list members are separated by spaces only
Heading_format=Topic
```

You can start a topic within a table only if you eliminate table structure for all tables in your document:

```
[Tables]
StripTables=Yes
```

See §8.5.3 [Converting table rows to topics and table cells to pop-ups](#) on page 262.

The format to which you assign property `Topic` becomes the title of the topic in WinHelp; see §8.8.3 [Configuring topic titles for WinHelp](#) on page 271.

You can create three kinds of WinHelp topics with **Mif2Go**:

Normal topics Usually begins with a heading in your document and continues to the next heading; appears in a full window when you jump to it.

Sliding topics Occurs in the middle of a normal topic in FrameMaker, but has to look like a separate topic in WinHelp; often a table or figure.

- Pop-up topics** Appears in a small window over the current normal topic when selected; use for definitions or glossary entries.
- Normal topics** A normal topic has properties such as the following:

```
[HelpStyles]
Heading 1=Topic Browse Key Contents
```

 See §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268 for information about these and other topic properties.
- Sliding topics** A sliding topic has property `Slide` in addition to `Topic` and any other properties. For example:

```
[HelpStyles]
TableTitle=Topic Slide Browse Key Contents
```

 Sliding topics are meant for embedded glossary terms, tables, or figures, where you want to lift something out of the middle of another topic and make that something a topic itself. A sliding topic does not end the previous topic (unless it too was a sliding topic), but just suspends the previous topic. A sliding topic ends at the next sliding topic, normal topic, or paragraph with a format to which you have assigned property `Resume`. For example:

```
[HelpStyles]
Body=Resume
```

 If the previous topic was a normal topic, that topic continues. This lets you handle tables, figures, or “lifts” within normal topics separately, without breaking the flow of the normal topic. If a sliding topic includes `Browse`, it appears in the browse sequence immediately after the enclosing normal topic.
- Pop-up topics** A pop-up topic is a single paragraph, usually not a heading. For example:

```
[HelpStyles]
Description=Topic Scroll NoXScroll NoTitle
```

 A topic designated for use as a pop-up cannot have a non-scrolling region, or it will pop up looking empty. Pop-up topics do not have titles; however, if you do not want pop-up topic text to appear in full-text search, you must assign property `NoTitle`.
 See also §8.9.2 [Configuring pop-up topics](#) on page 273.

8.8.2 Assigning properties to formats for topics and hotspots

Assign properties for topics and hotspots as follows:

```
[HelpStyles]
; style = key list, where list members are separated by spaces only
Format=StartingProperty FollowingProperty1 FollowingProperty2 ...
```

Properties for topics and hotspots are assigned to the following formats:

- Topic** Starting paragraph format; see §8.8.1 [Creating WinHelp topics](#) on page 267
- Hotspot** Delimiting character (or paragraph) format; see §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274.

[Table 8-2](#) shows which properties you can assign to formats as starting and following properties for topics and hotspots. [Table 8-3](#) shows the effects of the properties you assign.

Table 8-2 Starting and following format properties for topics and hotspots

Type	Starting	Following
Topic	PopContent	AKey Delete Key SpKey Suffix
	JumpTarget	AKey Contents Delete Key Local ¹ Resume SpKey Suffix
	Topic	AKey Browse Build Contents Delete Key Macro MakeRef NoScroll NoTitle NoXScroll Refer Scroll Slide SpKey Suffix TitleSuf Window XScroll
Hotspot	AKey	Delete Key Resume SpKey
	Delete	AKey Key Resume SpKey Suffix
	JumpHot	AKey Delete File Green Key Local Resume SpKey Suffix Uline Window
	Key	AKey Delete Resume SpKey
	MacroHot	AKey Delete Green Key Resume SpKey Uline
	ParaLink	Resume
	PopHot	AKey Delete File Green Key Local Resume SpKey Suffix Uline
	PopOver	AKey Delete Green ² Key ParaLink PrevRef Resume SpKey Suffix Uline ²
	Replace	AKey Key Resume SpKey Suffix
	Resume	Delete
	SpKey	AKey Delete Key Resume

¹ Within current topic only. ² Applied to cross references or hypertext links.,

Table 8-3 Effects of format properties on topics and hotspots

Property	Effect
AKey	Adds the topic title as an “A” footnote. See §8.11.2 Adding ALinks and KLinks with markers on page 285.
Browse	Includes the topic in the browse sequence; Mif2Go creates the “+” footnote. See §8.14 Creating browse sequences on page 292.
Build	Specifies a “build tag” for the topic, defined in the .hlp file, which is used to enable conditional compilation of different Help-file versions. The build tag name is in the [HelpTopicBuildStyles] section, as <i>format=buildtag</i> ; Mif2Go creates the “*” footnote.
Contents	Includes the topic (as a page, book, or both) in the WinHelp .cnt file produced by Mif2Go . See §8.13 Configuring contents for WinHelp on page 288.
Delete	Suppresses appearance in the output of text in the assigned format. <i>Tables and graphics anchored in a suppressed paragraph are retained</i> ; see §8.3.1.2 Suppressing unwanted paragraphs on page 253.
File	Directs the jump or pop-up to a topic in a different .hlp file. Mif2Go looks for the target in the file you specify in [HelpJumpFileStyles]. See §8.9.12.2 Designating interfile jumps and pop-ups on page 283.
Green	Makes a hotspot green and underlined. See §8.9.3.1 Configuring jump vs. pop-up hotspots on page 274.
Key	Adds the topic title as a keyword, so it appears in the WinHelp index. See §8.11.3 Adding related-topic keywords with formats on page 285.
Local	Produces a reference string that is local to the topic, so the same term can be used in more than one topic in the same .hlp file. See §8.9.12.1 Designating local-to-topic jumps and pop-ups on page 283.

Table 8-3 Effects of format properties on topics and hotspots (continued)

Property	Effect
Macro	Specifies an entry macro for the topic; this macro is run whenever the topic is selected. The macro text appears in the [HelpMacroStyles] section, as <i>format=macro</i> ; Mif2Go creates the “!” footnote. See §8.10 Invoking WinHelp macros on page 284.
MakeRef	Creates a unique reference tag for the current paragraph, to use as a pop-up or jump destination for a PrevRef paragraph; see §8.6.4 Displaying graphics in pop-ups for WinHelp on page 265.
NoScroll	Prevents the topic title from scrolling; overrides [HelpOptions]TitleScroll=Yes. See §8.8.3.2 Deciding whether to scroll titles on page 271
NoTitle	Prevents the topic title from being displayed; when applied to a pop-up topic, prevents the topic from appearing in full-text search.
NoXScroll	Overrides [HelpOptions]ExtendHelpNoScroll=No for pop-ups. See §8.8.3.2 Deciding whether to scroll titles on page 271.
ParaLink	Makes a full paragraph into a hotspot regardless of any character formats it includes. See §8.9.3 Creating hotspots for jumps and pop-ups in WinHelp on page 274.
PrevRef	Uses the previous unique reference (MakeRef) tag as a pop-up or jump destination for the whole paragraph or character span; if also PopOver, it is a pop-up. See §8.6.4 Displaying graphics in pop-ups for WinHelp on page 265.
Replace	Deletes the content, which is replace by the RTF code in [HelpReplacements].
Refer	Includes the topic name in a slightly modified form as a reference string for the topic. The modification consists of removing spaces and any characters other than letters, numbers, and underscores. Mif2Go creates the “#” footnote for you. Omit this property if the topic is accessed only from cross references or hypertext links, and from the Contents. See §8.9.11.5 Assigning properties to alternative jumps and pop-ups on page 281.
Resume	If the topic being processed is a sliding topic, causes that topic to end and the topic that preceded the sliding topic to continue. See §8.8.1 Creating WinHelp topics on page 267.
Scroll	Causes the topic title to scroll; overrides [HelpOptions]TitleScroll=No. See §8.8.3.2 Deciding whether to scroll titles on page 271.
Slide	Makes the topic a sliding topic. See §8.8.1 Creating WinHelp topics on page 267.
SpKey	Produces a keyword footnote designated with a different letter (neither “A” nor “K”) See §8.11.3 Adding related-topic keywords with formats on page 285.
Suffix	Differentiates character formats that use the same hotspot text but different references, by adding a suffix string; use the keyword Suffix for the formats, and add entries for those formats in [HelpSuffixStyles]format=Suffix text. See §8.9.11.5 Assigning properties to alternative jumps and pop-ups on page 281.
TitleSuf	Becomes a suffix to the topic title, to further define or categorize the topic; for example, for glossary terms. In [HelpStyles], use the keyword TitleSuf for the format, and add an entry for the format in [HelpTitleSufStyles]format=Suffix text. See §8.8.3.1 Categorizing titles on page 271.
Topic	Ends any prior topics and starts a new topic. Mif2Go creates the WinHelp topic start coding, a page break, and a title (\$) footnote, using the tagged text as the title. See §8.8.1 Creating WinHelp topics on page 267.
Uline	Underlines a hotspot without turning it green. See §8.9.11.4 Configuring hotspot appearance on page 281.
Window	Makes the topic appear in a specific window, defined in the .hbj file, when it is accessed from the Index or Find tabs, or from an ALink or KLink macro; but <i>not</i> when it is selected from a jump or from the Contents. The name of the window appears in the [HelpWindowStyles] section, as <i>format=Window</i> ; Mif2Go creates the “>” footnote. See §8.9.7 Specifying jumps to secondary windows in WinHelp on page 277.
XScroll	Overrides [HelpOptions]ExtendHelpNoScroll=Yes for pop-ups. See §8.8.3.2 Deciding whether to scroll titles on page 271.

8.8.3 Configuring topic titles for WinHelp

In this section:

§8.8.3.1 [Categorizing titles](#) on page 271

§8.8.3.2 [Deciding whether to scroll titles](#) on page 271

§8.8.3.3 [Fine-tuning title appearance](#) on page 272

8.8.3.1 Categorizing titles

To further define or categorize a topic—for example, for glossary terms—you can add a suffix to each topic title created with a particular format, by assigning the `TitleSuf` property to the format. For example:

```
[HelpStyles]
GlosTerm=Topic Key TitleSuf
```

To specify the text of the suffix, add an entry for the format in `[HelpTitleSufStyles]`:

```
[HelpTitleSufStyles]
GlosTerm=Suffix text
```

8.8.3.2 Deciding whether to scroll titles

<i>Keep title from scrolling</i>	<p>A title can either scroll with the text, or remained fixed in a no-scroll region at the top of the page. The default is for the title to remain fixed:</p> <pre>[HelpOptions] ; TitleScroll = Yes (title para scrolls with text), ; or No (fixed at top) TitleScroll=No</pre>
<i>Add to no-scroll region</i>	<p>You can extend the no-scroll region to include text that follows the title by setting the additional paragraphs to have the <i>Keep With Next</i> property in FrameMaker, and by specifying the following:</p> <pre>[HelpOptions] ; ExtendHelpNoScroll = No (default), ; or Yes (allow Keep With Next paras) ExtendHelpNoScroll=Yes</pre>
<i>Override no-scroll extension</i>	<p>To keep the <code>ExtendHelpNoScroll=Yes</code> setting but override it for particular paragraph formats, for each such format specify the following:</p> <pre>HelpStyles] YourFormat=NoXScroll</pre> <p>The <code>NoXScroll</code> property suppresses <i>Keep With Next</i> so that the no-scroll region is <i>not</i> extended for paragraphs in that particular format.</p> <p>Note: If a paragraph format is already listed under <code>[HelpStyles]</code>, just add the <code>NoXScroll</code> property to its current list of properties; do not repeat the format name.</p>
<i>Scroll title</i>	<p>To make titles scroll with the text, specify the following:</p> <pre>[HelpOptions] TitleScroll=Yes</pre>
<i>Override scrolling</i>	<p>You can still create a no-scroll region for such topics with this setting:</p> <pre>[HelpOptions] ExtendHelpNoScroll=Yes</pre>
<i>Pop-ups cannot be scrolled</i>	<p>Pop-up topics do not have scroll bars, so you might have to override the <code>TitleScroll</code> and <code>ExtendHelpNoScroll</code> settings. For each pop-up-topic paragraph format, add properties <code>Scroll</code> and <code>NoXScroll</code>. For example:</p>

```
[HelpStyles]
PopTopic=Topic Scroll NoXScroll
```

8.8.3.3 Fine-tuning title appearance

*Spaces and
indents*

You can allow or disallow spacing and indentation in topic titles:

```
[HelpOptions]
; TitleSpace = Yes (help title para can have space above/below),
;   or No
TitleSpace=No
; TitleIndent = Yes (help title para can have left/right indents),
;   or No
TitleIndent=No
```

*Hard returns in
titles*

By default, **Mif2Go** recognizes a hard return as the end of a topic title. If any topic titles continue past the first hard return, you can prevent this behavior:

```
[HelpOptions]
; HelpLineBreak = Yes (default, end topic title at hard return) or No
HelpLineBreak=No
```

8.9 Creating jumps and pop-ups for WinHelp

To invoke a jump or a pop-up in WinHelp, you click a *hotspot*: usually green underlined text. Clicking a hotspot causes one of the following:

- a *jump*, if the underline is solid: another topic replaces the topic in the current window
- a *pop-up*, if the underline is dotted: a smaller window pops up over the current window.

Mif2Go provides two systems for creating WinHelp jumps and pop-ups:

Current system: Based on FrameMaker cross references and hypertext links
Older system: Based on FrameMaker markers (see §8.9.11 [Configuring alternative jumps and pop-ups](#) on page 280)

Use only the current system, unless your project requires special-case jumps or pop-ups that cannot be produced with cross references or hypertext links. The older, alternative system is deprecated.

In this section:

- §8.9.1 [Identifying WinHelp jump destinations with FileIDs](#) on page 273
- §8.9.2 [Configuring pop-up topics](#) on page 273
- §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274
- §8.9.4 [Using cross references for jumps and pop-ups](#) on page 276
- §8.9.5 [Using hypertext links for jumps and pop-ups](#) on page 276
- §8.9.6 [Disallowing hypertext links for jumps and pop-ups](#) on page 277
- §8.9.7 [Specifying jumps to secondary windows in WinHelp](#) on page 277
- §8.9.8 [Specifying jumps to external files](#) on page 278
- §8.9.9 [Using the same content for both normal topics and pop-ups](#) on page 278
- §8.9.10 [Creating a glossary pop-up: an example](#) on page 280
- §8.9.11 [Configuring alternative jumps and pop-ups](#) on page 280
- §8.9.12 [Specifying the scope of alternative jumps and pop-ups](#) on page 283

See also:

- §7.7 [Jumping to secondary windows in Help systems](#) on page 224

§7.8 [Creating pop-up topics for Help systems](#) on page 225

§8.4.3 [Specifying cross-reference jump destinations](#) on page 260

§8.5.3 [Converting table rows to topics and table cells to pop-ups](#) on page 262

§8.6.4 [Displaying graphics in pop-ups for WinHelp](#) on page 265

8.9.1 Identifying WinHelp jump destinations with FileIDs

For a jump from a hypertext link, **Mif2Go** uses one of the following:

- the link text, for links within the same file
- the ObjectID, combined with the FileID or an *ad hoc* unique ID, for interfile links (see §5.3 [Identifying files and objects](#) on page 117).

Unless your project consists of only one file, with no cross references to other files, use the following default setting:

```
[HelpOptions]
; UseFileIDs = Yes (default, xrefs and ObjIDs)
; or No (single topic file)
UseFileIDs=Yes
```

When UseFileIDs=Yes, **Mif2Go** includes a FileID in the link code, so links do not get confused if a cross-reference number or ObjectID is not unique. **Mif2Go** assigns FileIDs to your FrameMaker files; see §5.3.4 [Working with Mif2Go FileIDs](#) on page 119.

*Keeping or
replacing FileIDs*

Suppose you have an existing WinHelp conversion project that uses the FileIDs you listed under [FileIDs] in your configuration file.

If you want to continue using the FileIDs listed in your configuration file, specify the following option:

```
[Setup]
; UseLocalFileID = No (default, use IDFile IDs)
; or Yes (use [FileIDs] here)
UseLocalFileID=Yes
```

If you want to switch to mif2go.ini FileIDs, stay with the default, UseLocalFileID=No; and update settings in any configuration sections that reference the original FileIDs, such as the [Graph*] sections.

8.9.2 Configuring pop-up topics

WinHelp pop-up topics (the contents of the small window that pops up) can be created just like any other topic. Assign the following properties to the paragraph format for a pop-up topic:

```
[HelpStyles]
Poptopicfmt=Topic Scroll NoXScroll
```

Suppose your document contains material that should appear in its original location in the print version, but would work better as a pop-up in the help version. For example, you might want “tips” to be pop-ups in WinHelp, even though in the print version they appear interspersed with regular text, or perhaps as sidebars.

Suppose the paragraph format you use for tips is called *Tip*:

1. Create another paragraph format to immediately follow every instance of *Tip*; call it (for example) *TipEnd*.
2. Make every instance of *TipEnd* conditional, for use only in the help version; for example, apply condition **HelpOnly**.

3. Do not include any text in the *TipEnd* paragraphs. Instead, use the numbering properties of *TipEnd* to make the word **Tip** appear automatically.
4. In the configuration file, specify the following settings for *Tip* and *TipAfter*:

```
[HelpStyles]
Tip = Topic Slide Scroll NoXScroll Makeref
TipEnd = PopOver Green Resume PrevRef
```

If you show condition **HelpOnly** when you use **Mif2Go** to generate WinHelp, the word **Tip** appears green and underlined in the WinHelp output. When a user clicks **Tip**, the text of the tip pops up. See §8.9.11.6 [Using unique references for jumps and pop-ups](#) on page 282.

See also:

- §8.8.1 [Creating WinHelp topics](#) on page 267
- §8.9.5.2 [Using alert or alerttitle markers for embedded pop-ups](#) on page 277
- §8.9.10 [Creating a glossary pop-up: an example](#) on page 280
- §34.1.1 [Using character formats to identify Help elements](#) on page 933

8.9.3 Creating hotspots for jumps and pop-ups in WinHelp

To create a hotspot, in FrameMaker apply a dedicated hotspot character format to either of the following:

- the text of a cross reference
- text that contains a hypertext link.

Note: If hotspot text is followed immediately by a space, when your WinHelp project is compiled the space disappears, even though it is present in the RTF output. The workaround is to make the space a hard space in FrameMaker.

In this section:

- §8.9.3.1 [Configuring jump vs. pop-up hotspots](#) on page 274
- §8.9.3.2 [Making a paragraph into a hotspot](#) on page 275
- §8.9.3.3 [Controlling hotspot appearance](#) on page 275
- §8.9.3.4 [Embedding hotspots in graphics for WinHelp](#) on page 275

See also:

- §7.8.1 [Understanding pop-up hotspots, links, and topics](#) on page 225

8.9.3.1 Configuring jump vs. pop-up hotspots

Pop-up hotspot For a pop-up hotspot, assign property PopOver to the hotspot character format:

```
[HelpStyles]
Hotspotfmt=PopOver
```

Specify PopOver as the first property to the right of the equals sign. Additional properties can follow PopOver; see §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268.

See also §8.9.11.3 [Creating alternative pop-ups](#) on page 281 for an older, alternative way to designate a pop-up hotspot.

Jump hotspot For a jump hotspot, you do not have to assign a property to the hotspot character format; jump is the default action for a hotspot.

See also §8.9.11.2 [Creating alternative jumps](#) on page 281 for an older, alternative way to designate a jump hotspot.

<i>Cross reference as a hotspot</i>	When you use a cross reference for a hotspot, make sure the hotspot character format is applied to the reference string (<\$paratext>) in the cross-reference definition. If another character format intervenes (for example, if your cross-reference format is defined as <Hotspotfmt><Bold><\$paratext></>), the PopOver property is turned off before it can be used.
<i>Hypertext link for a hotspot</i>	When you use a hypertext link to designate character-formatted text as a hotspot, do not place any other markers within the hotspot area.

8.9.3.2 Making a paragraph into a hotspot

To make an entire paragraph into a hotspot, if some of the text has a character format applied, you must assign the ParaLink property to the paragraph format. For example:

```
[HelpStyles]
; ParaLink can follow PopOver or appear alone; makes an entire para a
; hotspot for the first jump or popup ref, ignoring char formats
FigLOF=ParaLink
```

Do not place any markers within a paragraph that serves as a hotspot.

One additional property, Resume, can follow ParaLink; see §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268.

See also:

§5.10 [Creating hotspots for hypertext links](#) on page 138

8.9.3.3 Controlling hotspot appearance

By default, hotspot text is green and underlined. To leave the appearance of hotspot text *as is* in WinHelp, set the following option:

```
[HelpOptions]
;UseGreen = Yes (default) or No (remove green color from all links)
UseGreen=No
```

When UseGreen=No, hotspot character formats to which you have *not* assigned jump or pop-up properties in [HelpStyles] retain their original appearance. If you *have* assigned such properties, when UseGreen=No some automatically generated links appear underlined, but retain their original text color.

8.9.3.4 Embedding hotspots in graphics for WinHelp

To embed jump or pop-up hotspots in a graphic, you must first convert the graphic to a WinHelp “hypergraphic” with the Help Workshop Hotspot Editor, shed.exe, to create a .shg file from the graphic. Use the Hotspot Editor to set Hotspot IDs to the targets you want, typically names you inserted in your FrameMaker document as hypertext **newlink** markers (see §8.9.5 [Using hypertext links for jumps and pop-ups](#) on page 276), and specify whether each hotspot should be a pop-up or a jump.

After you create a .shg file, to make Mif2Go reference the hypergraphic, specify the following configuration settings:

```
[Graphics]
FilePaths=None
FileNames=Map

[GraphFiles]
myold.bmp=mynew.shg
```

8.9.4 Using cross references for jumps and pop-ups

<i>Cross references as jumps</i>	By default, Mif2Go converts all FrameMaker cross references into jumps for WinHelp, including interfile jumps where needed.
<i>Cross references as pop-ups</i>	To make a cross reference into a pop-up instead of a jump, assign property <code>PopOver</code> to the hotspot character format; see §8.9.3.1 Configuring jump vs. pop-up hotspots on page 274. The span of the hotspot is determined by the character format applied to the reference string.
<i>Disallowing cross references as pop-ups</i>	To disallow using cross references for pop-ups: <pre>[HelpOptions] ; NoXrefPopups = No (default, allow override to popup) or yes NoXrefPopups=Yes</pre>

8.9.5 Using hypertext links for jumps and pop-ups

By default, **Mif2Go** converts all FrameMaker hypertext links into jumps for WinHelp, including interfile jumps where needed. You can insert hypertext markers in your FrameMaker document to create additional jumps and pop-ups.

In this section:

§8.9.5.1 [Using openlink and gotolink markers](#) on page 276

§8.9.5.2 [Using alert or alerttitle markers for embedded pop-ups](#) on page 277

§8.9.5.3 [Using Type 11 markers for jumps or newlinks](#) on page 277

See also:

§34.1.2 [Using markers to add links and instructions](#) on page 935

8.9.5.1 Using openlink and gotolink markers

Use a FrameMaker **gotolink** or **openlink** marker to identify a hotspot, and type the reference string (the **newlink** name) as the marker text. Place the marker anywhere in the hotspot; the span of the hotspot is determined by the character format applied to text containing the marker. If no character format is applied, the entire paragraph becomes a hotspot; see §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274.

Note: Do not place any other markers within the hotspot area.

openlink The syntax for **openlink** is ambiguous. The FrameMaker *Hypertext* dialog says it is:

```
openlink filename:linkname
```

which could mean either:

```
openlink [filename:]linkname
```

(link name required, file name optional) which is how **gotolink** works, or:

```
openlink filename[:linkname]
```

(file name required, link name optional) which is how **openlink** works.

gotolink Use **gotolink** for targets within the same FrameMaker file:

```
gotolink refstring
```

Use **openlink** only for targets that are not in the current FrameMaker file:

```
openlink filename:refstring
```

Include the file extension with the name of the file that contains the target. For example:

```
openlink chap2.fm:redwoods
```

How to insert markers To insert **openlink**, **gotolink**, and **newlink** markers, see §34.1.2 [Using markers to add links and instructions](#) on page 935.

8.9.5.2 Using alert or alerttitle markers for embedded pop-ups

Instead of creating a pop-up topic, you can use an **alert** or **alerttitle** hypertext marker, and provide pop-up content in the text of the marker itself.

You designate a hotspot the same way as for other hypertext links; see §8.9.5.1 [Using openlink and gotolink markers](#) on page 276. Type the marker text in FrameMaker as a reference string consisting of the base file name, followed by **Alert**, followed by a number, starting with 0001 in each file. For example:

```
chap1.hlp Alert 0012
```

The pop-up text appears in the default format; you cannot specify a different format.

How to insert markers To insert **alert** and **alerttitle** markers, see §34.1.2 [Using markers to add links and instructions](#) on page 935.

8.9.5.3 Using Type 11 markers for jumps or newlinks

If your document includes old **Type 11** markers to support mid-topic jumps or context-sensitive help, you can continue using them. The marker can refer to a mid-topic link, act as a **newlink**, or be ignored.

```
[HelpOptions]
; MarkerType11 = Midtopic (default), Full (as newlink), None
MarkerType11=Midtopic
```

To use **Type 11** markers, assign one of the following characteristics to **MarkerType11**:

Midtopic	Markers are mid-topic links; start the marker text with the word midtopic .
Full	Markers are regular links; use the full marker text for the reference string (similar to newlink).
None	Markers are used for some other purpose; ignore Type 11 markers.

8.9.6 Disallowing hypertext links for jumps and pop-ups

If you do not want **Mif2Go** to use the hypertext links in your document for jumps or pop-ups, specify the following option:

```
[HelpOptions]
; UseHyperlinks = Yes (default) or No (ignore all hyperlinks)
UseHyperlinks=No
```

This setting causes all hypertext links to be ignored; instead the link locations are treated as plain text in WinHelp, and are not used as hotspots.

Note: This setting does not affect cross-reference hotspots and links.

8.9.7 Specifying jumps to secondary windows in WinHelp

Normally jumps make the topic you are jumping to appear in the main WinHelp window, replacing the previous content. If you want to make the new topic appear in a different window, assign the **Window** property to the hotspot character format. For example:

```
[HelpStyles]
JumpToExtra=JumpHot Green Window
```

Also add the `JumpHot` character format to `[HelpWindowStyles]`, specifying the name of the target window, exactly as specified in the `.hbj` file:

```
[HelpWindowStyles]
JumpToExtra=extra
```

This works for jumps, local and otherwise, but not for pop-ups.

When you specify a secondary window, you get only one instance; the next time you target that window, you replace its previous contents and leave it in place.

8.9.8 Specifying jumps to external files

Mif2Go produces links to external sources from hypertext markers in your document that contain the following types of hypertext “message” commands:

message URL

message openfile

When a **message openfile** link specifies an absolute path (which must start with a drive letter), **Mif2Go** removes any “file:///” URL prefix to the path, which is not needed in RTF. For example:

```
message openfile file:///g:/omnisys/ug/out/ugmif2go.pdf
```

becomes:

```
!EF('g:/omnisys/ug/out/ugmif2go.pdf')
```

Hypertext “message” commands are implemented in WinHelp with the `!EF` macro; see §8.10.1 [Using a hypertext marker to invoke a macro](#) on page 284.

*How to insert
markers*

To insert **message URL** and **message openfile** markers, see §34.1.2 [Using markers to add links and instructions](#) on page 935.

8.9.9 Using the same content for both normal topics and pop-ups

Suppose you want to create pop-ups that can be accessed from hotspots with varying text content; and suppose the text for each pop-up is already in use as a regular topic in your document. You could do either (or both) of the following:

- Copy the pop-up content to another file, and give the copy a different paragraph format, so you can assign it different properties in the configuration file. See §8.9.9.1 [Using a separate file for pop-ups](#) on page 278.
- Put the pop-up content in text insets, and thus avoid duplicating the content. See §8.9.9.2 [Using text insets for both pop-ups and normal topics](#) on page 279.

You could use the first method for cases where you want the text to be slightly different for topic and pop-up, and the second for true single-sourcing.

8.9.9.1 Using a separate file for pop-ups

To use the same content for both a normal topic and a pop-up, duplicating the text so you can edit topic and pop-up separately:

1. Create a new FrameMaker file, for example `Popups.fm`, and include it in your book.
2. Copy each pop-up topic section from its original FrameMaker file and paste it in `Popups.fm`.
3. If the heading format you use for these topics is (for example) *Head2*, assign normal topic properties to *Head2* in the configuration file; for example:

```
[HelpStyles]
Head2=Topic Browse Contents
```

4. For the topics in `Popups.fm`, change the format name to (for example) *Head2pop*, and in the configuration file assign to *Head2pop* the following properties:

```
[HelpStyles]
Head2pop=Topic Scroll NoXScroll NoTitle
```

This setting eliminates the no-scroll area (essential for pop-ups), and keeps the content out of full-text search, so that a search would find only the version in the original FrameMaker file. If the pop-up has more than one paragraph, do not assign properties in the configuration file to any but the first paragraph.

5. In `Popups.fm`, add a hypertext **newlink** marker (see §8.9.5 [Using hypertext links for jumps and pop-ups](#) on page 276) to the start of each *Head2pop*, and give each an appropriate alphanumeric identifier.
6. At the points in your document where each pop-up is to be referenced, add a hypertext **gotolink** marker with the proper identifier, and mark the hotspot span by applying a character format. For example, if you call the character format *Popup* (make sure it is in the catalog):

```
[HelpStyles]
Popup=PopOver Green
```

where *Green* is normal and recommended, but optional.

*How to insert
markers*

To insert **newlink** and **gotolink** markers, see §34.1.2 [Using markers to add links and instructions](#) on page 935.

8.9.9.2 Using text insets for both pop-ups and normal topics

To use the same content for both a normal topic and a pop-up, without duplicating text:

1. Create a new FrameMaker file, for example, `Popups.fm`, and include this file in your FrameMaker book.
2. Place each pop-up content chunk in a new tagged flow in `Popups.fm`, keeping the original paragraph format name; for example, *Head2*. You must give each flow tag a name; perhaps the same name as the identifier in the **newlink** marker for the pop-up. (Flow tag names are not restricted to single letters.)
3. Import each pop-up text inset into main text flow A in `Popups.fm`. Yes, you can do this, even if the insets are stored in the same file!
4. For each pop-up, do the following:
 - 4.1. Go to the pop-up text in its own named flow in `Popups.fm`, and cut the hypertext **newlink** marker.
 - 4.2. Go back to main flow A in `Popups.fm`, and paste the marker just before the head of the inset for that pop-up, in the same paragraph.
 - 4.3. Go to the place in the original FrameMaker file where the pop-up content was located, and replace that text with another import-by-reference of the pop-up from its own flow in `Popups.fm`. This time you do not want the hypertext **newlink** marker, and it will not be present.
5. Create a new configuration file, `Popups.ini`, that contains only the following setting (and any comments you want):

```
[HelpStyles]
Head2=Topic Scroll NoXScroll NoTitle
```

This ensures that when **Mif2Go** processes `Popups.fm`, you get the pop-up properties rather than the normal content properties for paragraph format *Head2*. You do not need an entry for *Head2* in `_m2winhelp.ini`.

The advantage of this method is simplified maintenance. You edit the pop-up content in only one place: its own text flow in `Popups.fm`.

8.9.10 Creating a glossary pop-up: an example

Suppose you want to use a glossary as a collection of pop-up topics; and suppose each glossary entry consists of a *GlossTerm* paragraph followed by one or more *GlossDef* paragraphs. You would assign the following properties to paragraph format *GlossTerm*:

```
[HelpStyles]
GlossTerm=Topic Scroll NoXScroll
```

*Cross-reference
pop-up link*

In the text of another topic, to reference a given term in the glossary, you could make an instance of the term into a cross reference to that same term in the glossary; the cross-reference format would use a character format called, for example, *PopText*:

```
<PopText><$paratext></>
```

*Hypertext pop-up
link*

Instead of using a cross reference, you could insert a **newlink** marker in the topic heading, and a corresponding **gotolink** marker in the hotspot; see §8.9.5 [Using hypertext links for jumps and pop-ups](#) on page 276. To insert **newlink** and **gotolink** markers, see §34.1.2 [Using markers to add links and instructions](#) on page 935.

*Hotspot
appearance*

If you do not want the term to look different from surrounding text in your FrameMaker document, you could set all the properties of the *PopText* format to “As Is”. In the configuration file you would assign the following properties to character format *PopText*:

```
[HelpStyles]
PopText=PopOver
```

This makes any instance of *PopText* a hot spot, which will appear green in WinHelp. Clicking that hotspot brings up the glossary term and its definition in a pop-up window.

8.9.11 Configuring alternative jumps and pop-ups

The material in this section discusses methods and settings that are deprecated, except for certain situations where simple FrameMaker cross references or hypertext links do not suffice.

In this section:

§8.9.11.1 [Understanding alternative jumps and pop-ups](#) on page 280

§8.9.11.2 [Creating alternative jumps](#) on page 281

§8.9.11.3 [Creating alternative pop-ups](#) on page 281

§8.9.11.4 [Configuring hotspot appearance](#) on page 281

§8.9.11.5 [Assigning properties to alternative jumps and pop-ups](#) on page 281

§8.9.11.6 [Using unique references for jumps and pop-ups](#) on page 282

8.9.11.1 Understanding alternative jumps and pop-ups

To create an alternative jump or pop-up, **Mif2Go** produces a *reference string* from the hotspot text, which must match the reference string of the target topic. When you use a character (or paragraph) format to identify a jump or pop-up, the hotspot text string must be the same as the text of the target paragraph; **Mif2Go** depends on this equivalence. Because a reference string contains the full text of the target paragraph, and the acceptable length of a WinHelp reference string is very limited, this method is useful only for short pop-ups or jumps to short headings.

8.9.11.2 Creating alternative jumps

To create a jump without using a cross reference or a hypertext link, specify the following properties:

- Assign `JumpHot` to the hotspot character format.
- Assign `JumpTarget` (instead of `Topic`) to the topic heading format.

Each must be the first property to the right of the equals sign. For example:

```
[HelpStyles]
Hotspotfmt=JumpHot
Althead=JumpTarget
```

Additional properties can follow `JumpHot` or `JumpTarget`; see §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268. For example:

- `JumpHot` can have `Window`, `File` or `Local`, `Green`, and `Uline`.
- `JumpTarget` can have `Contents`.

8.9.11.3 Creating alternative pop-ups

To create a pop-up without using a cross reference or a hypertext link, specify the following properties:

- Assign `PopHot` (instead of `PopOver`) to the hotspot character format.
- Assign `PopContent` (instead of `Topic`) to the topic heading format.

Each must be the first property to the right of the equals sign. For example:

```
[HelpStyles]
Hotspotfmt=PopHot
Altstuff=PopContent
```

Additional properties can follow `PopHot` or `PopContent`; see §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268. For example:

- `PopHot` can have `File` or `Local`, `Green`, and `Uline`.
- `PopContent` can have `Local` (within current topic only).

8.9.11.4 Configuring hotspot appearance

In FrameMaker You can define a hotspot by applying only a color in FrameMaker; however, by default **Mif2Go** ignores colored text. To make **Mif2Go** recognize colored text as a hotspot:

```
[HelpOptions]
; UseHyperColor = No (default) or Yes (treat any non-black as hyper)
UseHyperColor=Yes
```

In WinHelp You can specify a pop-up hotspot's appearance in WinHelp by assigning either of the following properties to the hotspot character format:

<code>Green</code>	Makes the hotspot green and underlined.
<code>Uline</code>	Underlines the hotspot, without turning it green.

For example:

```
[HelpStyles]
PopReg=PopHot Green
PopNotGreen=PopHot Uline
```

8.9.11.5 Assigning properties to alternative jumps and pop-ups

If the hotspot text is the same as the target's reference string, apply a character format, and assign that format property `JumpHot` for a jump or `PopHot` for a pop-up. For example:


```
[HelpStyles]
GlossJump=JumpHot Green Key
PopUpDef=PopHot Green Uline
```

If the target is a topic, its reference string is already coded: it is the same as the topic title.

If the target is not a topic, identify the target's reference string by applying a character format coded `JumpTarget` for jumps or `PopContent` for pop-ups. A `JumpTarget`, sometimes called a *mid-topic jump*, can also have the word `Contents` in its definition, so that it appears in the `.cnt` file; see §8.13 [Configuring contents for WinHelp](#) on page 288:

```
[HelpStyles]
GlossTerm=JumpTarget Key Contents
LocalPop=PopContent
```

If the target is a fixed reference regardless of the text to which the style is applied, add `Refer` to the character format coding, and add a `[HelpRefStyles]` entry for the character format. For example, suppose you want to apply character format *Fruits* to make “apple”, “orange”, and “pear” hotspots, but in every case jump to the topic “fruit salad”:

```
[HelpStyles]
Fruits=JumpHot Refer Green

[HelpRefStyles]
Fruits=fruitsalad
```

If the hotspot text is the same for several items but you want a `JumpHot` or `PopHot` to go to different references, you must use four character formats, and add `Suffix` to the coding for each. For example, suppose at one point you want to mark *Apple* to jump to *Apple Computer*, and at another place in the same file you want to mark *Apple* to jump to *Apple Records*. You could define character formats *Comp*, *CompRef*, *Record*, and *RecRef*, and code them as follows:

```
[HelpStyles]
Comp=JumpHot Suffix Green
CompRef=JumpTarget Suffix
Record=JumpHot Suffix Green
RecRef=JumpTarget Suffix

[HelpSuffixStyles]
Comp=comp
CompRef=comp
Record=rec
RecRef=rec
```

You would apply *CompRef* and *RecRef* to the word *Apple* in targets *Apple Computer* and *Apple Records*, respectively. Then wherever else you apply *Comp* to *Apple*, you get a jump to *Apple Computer*; and wherever else you apply *Record* to *Apple* you get a jump to *Apple Records*.

8.9.11.6 Using unique references for jumps and pop-ups

You can use a pair of format properties to do the following:

- assign a unique reference tag to the destination paragraph
- assign a reference to that tag to a hotspot.

For example, the following settings would make each paragraph with format *Sidebar* a pop-up, and the next following *SeeSidebar* format a corresponding hotspot:

```
[HelpStyles]
; MakeRef creates a unique reference tag in the current paragraph.
; PrevRef uses the previous unique reference (MakeRef) tag as a jump
; destination for the whole paragraph; if also PopOver, a popup.
```

```

Sidebar=Topic Slide Scroll NoXScroll MakeRef
SeeSidebar=PopOver Green Resume PrevRef

```

The hotspot must *follow* the destination in your FrameMaker document, or the PrevRef property will not take you there when you click the hotspot.

Note: PrevRef should not be assigned to a format that already has some other hotspot-creating property (JumpHot, PopHot, MacroHot, a cross reference, or a hypertext **gotolink** or **openlink**).

See §8.8.2 [Assigning properties to formats for topics and hotspots](#) on page 268

8.9.12 Specifying the scope of alternative jumps and pop-ups

If all the jump and pop-up link references in your document are unique, and they are all in the same file as their corresponding hotspots, you do not have to specify a scope.

In this section:

§8.9.12.1 [Designating local-to-topic jumps and pop-ups](#) on page 283

§8.9.12.2 [Designating interfile jumps and pop-ups](#) on page 283

8.9.12.1 Designating local-to-topic jumps and pop-ups

Normally, a term used as a reference string must be unique in each .hlp file; you can jump to a reference string from any place in the same .hlp file. However, there is a way to use the same reference string in every topic.

If you need to use a term that appears in many places in the .hlp file, but only once in each topic, you can assign property Local to a jump or pop-up format to produce a local reference string. For example, you might have a catalog in which each item is in a separate topic, but every item needs a link within its topic to its part number, price, and so on.

Assign property Local to both the hotspot format and its target format. For example:

```

[HelpStyles]
Specs=JumpTarget Local
JSpecs=JumpHot Local
Diagram=PopContent Local
PDiag=PopHot Local

```

8.9.12.2 Designating interfile jumps and pop-ups

For jumps and pop-ups to topics in a different .hlp file, assign property File to the JumpHot or PopHot character format. For example:

```

[HelpStyles]
JElsewhere=JumpHot Green File
POther=PopHot File

```

Also assign the name of the target file to the hotspot character format. For example:

```

[HelpJumpFileStyles]
JElsewhere=distant.hlp
POther=distant.hlp

```

Also specify FileIDs; see §8.9.1 [Identifying WinHelp jump destinations with FileIDs](#) on page 273.

8.10 Invoking WinHelp macros

You can invoke a WinHelp macro from a hotspot. You can use this feature to create a jump to an external location, or to run another application from within WinHelp.

To create a macro hotspot, apply a character format to the hotspot text in FrameMaker, the same way you would for a jump or a pop-up; see §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274.

In this section:

§8.10.1 [Using a hypertext marker to invoke a macro](#) on page 284

§8.10.2 [Assigning a hotspot property to invoke a macro](#) on page 284

8.10.1 Using a hypertext marker to invoke a macro

The simplest way to create a jump to an external file is to insert a FrameMaker **Go to URL** hypertext marker where you want the link. For example, to run an .avi video clip:

```
message URL yourname.avi
```

Indicate the hotspot area with a character format that includes the marker; see §8.9.3 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 274. **Mif2Go** converts the marker text to produce the required WinHelp macro invocation:

```
!EF('yourname.avi')
```

Clicking the hotspot should display the .avi file.

See also:

§8.9.8 [Specifying jumps to external files](#) on page 278

§34.1.2 [Using markers to add links and instructions](#) on page 935

8.10.2 Assigning a hotspot property to invoke a macro

If you reference the same external file from several places in your document, you can dedicate a hotspot character format to this purpose. You assign property `MacroHot` to the character format, and provide a definition for the macro. For example:

```
[HelpStyles]
ShowAvi=MacroHot

[HelpMacroStyles]
ShowAvi=EF('yourname.avi')
```

The `MacroHot` property works the same way as `JumpHot` (see §8.9.11.2 [Creating alternative jumps](#) on page 281), except that instead of using the marked text as the reference string for a jump, **Mif2Go** provides a predefined WinHelp macro. For example:

```
[HelpStyles]
FarTarget=MacroHot

[HelpMacroStyles]
; Topic Macro and MacroHot have a required macro content
FarTarget=EF(http://www.omsys.com/)
```

Mif2Go supplies the leading “!” for the macro invocation.

8.11 Creating related-topic links in WinHelp

You can create ALink target topics in WinHelp by assigning “A” footnotes to formats, and provide links to those topics by embedding WinHelp ALink macros in the text of your document. Or, you can use markers for both ALink jumps and ALink-list targets.

Mif2Go constructs KLinks for WinHelp 4 automatically, from FrameMaker index markers. You can add other “K” footnotes that do not appear in the **Mif2Go**-generated index, either with formats or with markers.

Advantages of markers

Using markers for related-topic links has some advantages:

- The same markers work for ALinks and KLinks in all other **Mif2Go**-generated Help systems that support related-topic links.
- **Mif2Go** creates the WinHelp ALink or KLink macros for you when you use markers.

In this section:

- §8.11.1 [Understanding KLink limitations](#) on page 285
- §8.11.2 [Adding ALinks and KLinks with markers](#) on page 285
- §8.11.3 [Adding related-topic keywords with formats](#) on page 285
- §8.11.4 [Inserting WinHelp macros for ALink jumps](#) on page 286

See also:

- §7.6 [Providing related-topic links for Help systems](#) on page 219

8.11.1 Understanding KLink limitations

Although WinHelp 4 nominally supports KLinks, the following restrictions apply:

- Only the first index term in a KLink jump results in an active link, unless another .hlp file is linked in the .cnt file for your project. For example:

```
:Link other.hlp
:Index title=other.hlp
```
- When two index terms in the same KLink jump reference the same topic, the link appears twice in the KLink list.
- Multi-level index terms do not result in links.

8.11.2 Adding ALinks and KLinks with markers

To create ALinks and KLinks in WinHelp with markers, use the methods described in the following sections:

- §7.6.4.1 [Adding related-topic link keywords via markers](#) on page 221
- §7.6.5 [Adding ALink and KLink jumps in FrameMaker](#) on page 222

8.11.3 Adding related-topic keywords with formats

You can mark text in your document to produce the “A” footnotes used by ALinks, “K” footnotes for KLinks, or any other kind of WinHelp keyword footnote. You assign a related-topic keyword property to a paragraph or character format, along with other properties.

Keyword properties produce the following:

- AKey an “A” footnote, for an ALink; see §7.6.2 [Understanding how ALinks work](#) on page 220

- Key** a “K” footnote, for a KLink; see §7.6.3 [Understanding how KLinks work](#) on page 221
- SpKey** a “special” footnote, designated by a letter other than “A” or “K”, for a separate index that is searchable only when WinHelp is called from a program.

You can apply these properties in any of the following ways:

[Assign a keyword property to a paragraph format](#)

[Assign a keyword property to a character format](#)

[Assign a keyword property to hidden content](#)

[Assign a “special” keyword property to a format](#)

Assign a keyword property to a paragraph format

To create a “K” footnote for each level-2 topic heading (for example), you could assign property **Key** to the topic-heading paragraph format:

```
[HelpStyles]
Heading 2=Topic Browse Key
```

Assign a keyword property to a character format

To designate keywords in topic text, apply a character format to the relevant text, and assign a keyword property to the character format. For example, if the name of an “A” footnote subject appears as a word in topic text, you can apply a special character format to that word, and assign the **AKey** property to the format:

```
[HelpStyles]
Related=AKey
```

Assign a keyword property to hidden content

To add to a paragraph a footnote that does *not* appear in topic text:

1. Place the footnote text at the end of the paragraph, after the last period.
2. Apply a special character format to the footnote text.
3. Assign properties **AKey** (for example) and **Delete** to the character format:

```
[HelpStyles]
Atag=AKey Delete
```

The **Atag** text would be put into an “A” footnote, but would not appear in the topic.

Assign a “special” keyword property to a format

To create a “special” footnote, assign property **SpKey** to a format, and also assign a letter, other than A or K, to the same format. For example:

```
[HelpStyles]
xlink=SpKey

[HelpKeywordStyles]
; SpKey requires a key letter (A..Z, except K and A)
xlink=X
```

8.11.4 Inserting WinHelp macros for ALink jumps

You can create an authorable button (or just a hotspot) for “Related Topics” (usually either in the no-scroll region at the top, or at the end of the topic text), and provide a WinHelp ALink macro that lists the subject(s) to which you want link(s).

To create a jump to one or more ALink lists, insert a WinHelp ALink macro in text wherever you want a jump to appear. For example, to add a “Related Topics” button:

```
{button Related Topics:AL(subject1,subject2,...)}
```

8.12 Configuring index entries for WinHelp

Mif2Go converts FrameMaker index entries into WinHelp “K” footnotes. You can add to the WinHelp index other text that is not part of the FrameMaker index, by assigning properties to formats or by inserting markers.

In this section:

- §8.12.1 [Designating index level separators](#) on page 287
- §8.12.2 [Eliminating duplicate keywords](#) on page 287
- §8.12.3 [Keeping or discarding “See also” entries](#) on page 288
- §8.12.4 [Using FrameMaker Index markers](#) on page 288

See also:

- §7.5 [Configuring index entries for Help systems](#) on page 211

8.12.1 Designating index level separators

Commas in FrameMaker index entries are treated as level separators in WinHelp indexes, even though they serve only a grammatical function in your FrameMaker document. However, **Mif2Go** breaks an index entry at a comma (or at any other level separator) only when there is at least one more entry that is an exact match up to the comma. The “one more entry” can be an entry generated by **Mif2Go**, depending on the setting for `DisambiguateIndex`; see §8.12.2 [Eliminating duplicate keywords](#) on page 287.

To have **Mif2Go** treat commas in FrameMaker index entries as regular characters instead of index level separators, specify the following option:

```
[HelpOptions]
; IdxColon = No (default, allow colon and comma as level delimiters)
; or Yes (use only colon as delimiter, treat comma as regular text)
IdxColon=Yes
```

You must also edit the WinHelp project file, *yourdoc.hpj*, to specify this option:

```
[OPTIONS]
INDEX_SEPARATORS=":"
```

Use Notepad or any other plain-text editor.

8.12.2 Eliminating duplicate keywords

When a first-level index entry has second-level entries under it, to avoid repeats of the same topic, set `DisambiguateIndex=Topic`:

```
[HelpOptions]
; DisambiguateIndex = Yes (default, always write first-level keys),
; Strip (no first-level keys), Topic (only write first instance of
; a first-level key in each topic), No (only write first in doc)
DisambiguateIndex=Topic
```

The `DisambiguateIndex` options work as follows:

Topic	Prevents duplication by suppressing repeated index markers within a topic. When you have the same index marker in two or more places, and pick that item in the WinHelp index, you get a dialog with a list of all the places the item is referenced. If the same marker occurs twice in the same topic, you get two identical entries for that topic in the dialog list.
-------	---

Yes	Prevents duplication by eliminating repeated first-level headings in the file (not just in the topic). When you have second-level topics under the same first-level topic, and you click the first-level heading, you get duplication. You need only <i>one</i> of the first-level headings in the file to make the index work right (avoiding a WinHelp defect).
No	Generates a first-level heading only for the first of its second-level topics.
Strip	Eliminates generated first-level headings, so that only explicit headings remain.

8.12.3 Keeping or discarding “See also” entries

You can choose to discard “See also” entries in the index; the default is to keep them:

```
[HelpOptions]
; NoSeeAlso = No (default, leaves See also entries in index)
;   or Yes (removes them)
NoSeeAlso=No
```

8.12.4 Using FrameMaker Index markers

By default, **Mif2Go** omits the ObjectIDs of FrameMaker **Index** markers, unless you are converting the FrameMaker index. To restore use of ObjectIDs for **Index** markers:

```
[HelpOptions]
; KeepIXMarkerIDs = No (default, keep only if [Setup]UseFrameIX) or
;   Yes (always keep Unique ObjectIDs for Index markers)
KeepIXMarkerIDs=Yes
```

8.13 Configuring contents for WinHelp

WinHelp 4 uses a contents, or .cnt, file for each .hlp file. When you invoke Help, the system brings up a contents page, where you see little book icons (headers) and page icons (topics). If you click a book icon, it expands to show you the books and pages it contains; if you click a page icon, the associated topic is displayed in a Help window.

In this section:

§8.13.1 [Naming and configuring Help files and titles](#) on page 288

§8.13.2 [Specifying heading formats and levels for contents](#) on page 289

§8.13.3 [Assembling WinHelp contents from the command line](#) on page 291

See also:

§7.3.5 [Modifying contents or index production for WinHelp](#) on page 208

§7.4 [Configuring contents entries for Help systems](#) on page 209

8.13.1 Naming and configuring Help files and titles

Mif2Go creates a WinHelp contents file for you, unless you specify no contents. Whether the contents file is created from a single topic file or from multiple files, and how the contents file and Help file are named, depend on the following settings:

```
[HelpContents]
; the optional .cnt file for HelpVer 4 is always named after the rtf
; CntType = None, Full (single file), or Body (headings, topics only)
; the Body type is used when combining .cnt files in a .bat file
CntType=Full
; CntBase = helpfile.hlp (default is rtfname.hlp; specify for Body)
```



```

CntBase=myfile.hlp
; CntName = helpfile.cnt (default is rtfname.cnt; specify for Body)
CntName=myfile.cnt
; CntStartFile = helpfile.bct (default is to use CntBase and CntTitle)
CntStartFile=myfile.bct
; CntTitle = Title for Contents (for Full .cnt)
;CntTitle=Project Name
; CntTopic = starting topic for .hpj (default: book or chapter name)
;CntTopic=myfile
; CntTopHead = 1 Text for Optional Top Head (Full .cnt, for a H1)
;CntTopHead=1 Book Title

```

If you have only one topic file in your Help project, in the [HelpContents] section leave the default value, CntType=Full; **Mif2Go** creates the .cnt file for you.

If you have more than one topic file in your Help project, set CntType=Body; **Mif2Go** creates part of the .cnt file for each topic file, and also produces a file named after your topic file, with extension .bct, which contains only the header and topic lines. If you specify CntType=Body, you can also specify the name of the contents file and the base name of the help file. If you do not specify a base name, **Mif2Go** uses your topic file name for the help file base name.

Alternatively, for multiple topic files you can set the first topic file to CntType=Full, so it contains the Base, Title, and Top Head (if any). Or you can just prepare the first bit of the final .cnt separately, in a text editor.

Use CntTitle to specify the text of the title for the contents file; to specify a starting topic, set CntTopic. If you do not specify a value for CntTitle, **Mif2Go** uses the help file title (in the .hpj file). The value you specify (if any) for CntTopHead is added before the actual .cnt entry lines.

You can also create a .cnt heading for a topic file even though that heading is not in the topic file itself:

```

[HelpContents]
CntType=Body

[BctFileHeads]
myfile=Text for optional top head

```

8.13.2 Specifying heading formats and levels for contents

In this section:

§8.13.2.1 [Understanding WinHelp contents level numbers](#) on page 289

§8.13.2.2 [Listing topics for contents with and without subheadings](#) on page 290

§8.13.2.3 [Using different names in contents for heading and topic](#) on page 290

§8.13.2.4 [Renaming or eliminating the contents “Overview” topic](#) on page 290

§8.13.2.5 [Referencing multiple help files from contents](#) on page 291

§8.13.2.6 [Displaying contents targets in the main window](#) on page 291

8.13.2.1 Understanding WinHelp contents level numbers

In a WinHelp .cnt file, each heading line begins with a level number, 1 to 9, followed by the text to display. Each topic line includes the same, and adds an equals sign, followed by the reference string for the topic to be displayed. **Mif2Go** produces these lines for each format value in [HelpStyles] that starts with Topic and includes Contents. **Mif2Go** determines the type of line from the [HelpCntStyles] section where *formatname*=H for top-level headings, *formatname*=T2 for second-level topics, or *formatname*=B2 to

create a second-level heading with a third-level topic of the same name immediately following:

```
[HelpStyles]
Heading 1=Topic Contents
[HelpCntStyles]
; format = H (heading), T (topic), or B (both), + level (1..9), as in:
; Heading 2=B3 which creates both a level 3 head and a level 4 topic
; format V adjusts itself to be either T or B, depending on subheads
; all formats here must be listed in [HelpStyles] with Contents set
ChapName=H1
Heading 1=B2
```

8.13.2.2 Listing topics for contents with and without subheadings

Suppose you use the same format for topics with subheadings and for topics without subheadings. The possibilities include:

- All “book” entries
- All “page” entries
- Mixed “book” and “page” entries
- No subheadings? You get “page” entries
- Force “book” entries

- | | |
|---|--|
| <i>All “book” entries</i> | To list all such topics as “books” in the contents file, assign to the format a B (“both”) level: one of [HelpCntStyles] properties B1 through B9. Mif2Go creates a “book” contents entry and a subordinate “page” entry for each. |
| <i>All “page” entries</i> | To list all such topics as “pages” in the contents file, assign to the format a T (“topic”) level: one of [HelpCntStyles] properties T1 through T9. Mif2Go creates a “page” contents entry for each. |
| <i>Mixed “book” and “page” entries</i> | To list topics without subheadings as “pages”, and topics with subheadings as “books”, assign to the format a V (“variable”) level: one of [HelpCntStyles] properties V1 through V9. You can adjust the level numbers to fit the way you use your headings; this might take a bit of experimenting to get the effect you want. |
| <i>No subheadings? You get “page” entries</i> | When you assign V1 through V9, Mif2Go determines whether the format should be a heading or a topic, based on the existence of subheadings. However, a defect in WinHelp causes V entries without subheadings to be treated as though they were T. For example, a chapter heading for a single-topic chapter would appear as a “page” entry under the preceding chapter. |
| <i>Force “book” entries</i> | To list topics without subheadings as “books” (for example, if you have a single-topic chapter), you must assign a B level to the heading format instead of V. |

8.13.2.3 Using different names in contents for heading and topic

When you code a format to create both a heading and a topic, the default is to use the same text for both. To give the topic a more generic name, such as Overview or Summary, in [HelpContents] set CntBStyleText=other topic name:

```
[HelpContents]
; CntBStyleText = text to use for topics created as "B" HelpCntStyles
CntBStyleText=Overview
```

8.13.2.4 Renaming or eliminating the contents “Overview” topic

When a heading has subheadings under it (so that it becomes a “book” in WinHelp), WinHelp provides no way to get from the contents entry to any text that comes after the

heading and before the first subheading. Therefore **Mif2Go** adds a dummy topic “page” called (by default) “Overview” to permit access to this otherwise orphaned text.

You can change the name of this dummy topic to something other than “Overview”; for example, to name it “Introduction” instead:

```
[HelpContents]
CntBStyleText=Introduction
```

If you do not want to provide access to any text in this area, you can change the setting for the heading from V to H, keeping the same level number; for example:

```
[HelpCntStyles]
Heading1=H1
```

Then there would be no “Overview” for text that immediately follows a *Heading1* paragraph. You would be able to access the text between that heading and the next only as part of a browse sequence; see §8.14 [Creating browse sequences](#) on page 292.

8.13.2.5 Referencing multiple help files from contents

The lines in a .cnt file that are clickable links contain equals signs (“=”). The part to the right of the equals sign is the *reference string*. For example, suppose you are making a contents file called merged.cnt, which is your master contents file modified by inclusion of lines such as the following:

```
:include someother.cnt
```

The reference strings in someother.cnt must specify that the topics they identify are in someother.hlp, else they will be looked for in merged.hlp when they are accessed via merged.cnt. This is done for you if you specify the following, in each configuration file for a secondary .hlp file; it is not needed for the master .hlp file:

```
[HelpContents]
; AddCntFileName = No (default) or Yes (add to topic ref strings)
AddCntFileName=Yes
```

8.13.2.6 Displaying contents targets in the main window

If you are using multiple windows (secondary windows), to force material called from the contents into the main window, set AddCntWindowName=Yes; otherwise that material goes into the last secondary window used:

```
[HelpContents]
; AddCntWindowName = No (default)
; or Yes (add def >main to ref strings)
AddCntWindowName=No
```

Specify the name of the primary window, if it is not main:

```
[HelpContents]
; CntMainWindow = name for primary window in .cnt if not "main"
; CntMainWindow=myhelpmain
```

8.13.3 Assembling WinHelp contents from the command line

When you have generated all topic files for WinHelp, so that you have all the .bct files, you can put them together to form the full .cnt file; either with a text editor, or with the Windows copy command. For example:

```
copy intro.cnt+chap1.bct+chap2.bct+appx.bct mybook.cnt
```

If you have set up a batch file with other DCL conversion commands, add the **copy** line after the last **dcl** line and before the **hclw** line.

See §37 [Converting via DCL](#) on page 995.

8.14 Creating browse sequences

WinHelp supports *browse sequences* among topics, and provides a pair of browse buttons, << and >>, on the toolbar. Each topic can belong to only one browse sequence; if you want branching, you must use a jump to get to the first topic in the branch.

In this section:

§8.14.1 [Setting up an automatic browse sequence](#) on page 292

§8.14.2 [Specifying browse numbers](#) on page 292

§8.14.3 [Setting up multi-file browse sequences](#) on page 293

§8.14.4 [Setting up branching browse sequences](#) on page 293

8.14.1 Setting up an automatic browse sequence

The easiest way to provide a browse sequence is to use the WinHelp auto-browse feature, which strings together all the topics in your document in their order of appearance in the RTF files listed under [FILES] in the .hlp file.

To set up an automatic browse sequence:

```
[HelpBrowse]
; AutoBrowse = No (default) or Yes (an "auto" browse
; sequence is generated with topics in the order shown
; under [FILES] in the .hlp, in the order present within
; each file; no numbers are used, and Prefix is "auto").
AutoBrowse=Yes
```

When AutoBrowse=Yes, you do not have to maintain browse numbers in the configuration file. However, be aware of the following:

- Auto-browse works only when you have a single browse sequence.
- The browse sequence includes *every* topic, even pop-up topics.

8.14.2 Specifying browse numbers

You can specify a starting browse number in the configuration file; if you have more than one file in your project, you can specify a starting number for each. You can also specify a starting browse number in the document itself, in a configuration marker (see §33.2.2 [Overriding settings with configuration markers](#) on page 921).

To activate browse sequences for topic titles, in [HelpBrowse] specify TitleBrowse=Yes. Specify the interval to use between browse numbers as Step=5, or just Step=1 if you will never need to interpolate any new ones by hand. Specify the number of digits to use so that you have enough numbers available: Digits=3 allows for 999 topics, Digits=4 allows 9,999. Specify the starting browse number as Start=N, and the browse prefix as Prefix=XX. The default values are as follows:

```
[HelpBrowse]
; these defaults are for all files processed
; override as needed in individual filename.ini files
; or using configuration markers in the documents themselves
Step=5
Digits=4
; make sure each RTF file has a different Start value
; allowing room for the numbers used in the earlier files
Start=5
Prefix=HLP
```

8.14.3 Setting up multi-file browse sequences

If your help project contains more than one topic file, and you want to be able to browse from one to the next (as users generally expect), you will need to specify a different Start number for each file. You can do this in three places:

- in the [BrowseStart] section
- in a configuration marker in the FrameMaker file (see §33.2.2 [Overriding settings with configuration markers](#) on page 921)
- in a configuration file specifically for each FrameMaker file (see §33.1 [Using a different configuration for selected files](#) on page 919).

To place the start numbers in the [BrowseStart] section:

```
[BrowseStart]
; overrides the [HelpBrowse] Start above for the file named
; filename (no extension) = start number
```

To use a *file-specific* configuration file, create a plain text file with the same base name as the RTF file you are producing, but with extension .ini. All it needs to contain is:

```
[HelpBrowse]
Start=nnnn
```

Allow enough numeric room between successive Start settings to accommodate the maximum number of topics you might have in each file. You can include any settings specific to a particular file in such a configuration file; they override the corresponding settings in the m2winhelp.ini file.

8.14.4 Setting up branching browse sequences

For branching browse schemes, use a different prefix for each branch. You can specify a prefix in the configuration file, in a marker, or in a specific configuration file, as for start numbers (see §8.14.3 [Setting up multi-file browse sequences](#) on page 293):

```
[BrowsePrefix]
; overrides the [HelpBrowse] Prefix for the file named
; filename (no extension) = prefix string
; is overridden itself by usage in [HelpBrowsePrefixStyles]

[HelpBrowsePrefixStyles]
; Topic Browse can have an optional prefix
```


9 Generating Microsoft HTML Help

This section addresses issues that are specific to creating Microsoft HTML Help. HTML settings described in section 13 and sections 18 through 34 apply also. Topics include:

- §9.1 [Understanding how Mif2Go produces HTML Help](#) on page 295
- §9.2 [Understanding why Unicode is not the answer](#) on page 296
- §9.3 [Setting up an HTML Help project](#) on page 297
- §9.4 [Customizing HTML Help display features](#) on page 302
- §9.5 [Creating pop-ups for HTML Help](#) on page 305
- §9.6 [Creating links and hypertext jumps in HTML Help](#) on page 307
- §9.7 [Creating related-topic links for HTML Help](#) on page 309
- §9.8 [Using secondary windows in HTML Help](#) on page 317
- §9.9 [Generating contents and index for HTML Help](#) on page 319
- §9.10 [Converting generated files for HTML Help](#) on page 325
- §9.11 [Providing full-text search \(FTS\) for HTML Help](#) on page 326
- §9.12 [Setting up CSH for HTML Help](#) on page 326
- §9.13 [Generating HTML Help in non-Western languages](#) on page 331
- §9.14 [Compiling and testing HTML Help](#) on page 333
- §9.15 [Mapping and merging CHM files](#) on page 336

See also:

- §7 [Producing on-line Help](#) on page 199

To determine which configuration settings will produce the appearance and functionality you want, see also:

- §13 [Converting to HTML/XHTML](#) on page 423
- §18 [Splitting and extracting files](#) on page 585
- §21 [Mapping text formats to HTML/XML](#) on page 645
- §22 [Setting up CSS for HTML](#) on page 681
- §23 [Including graphics in HTML](#) on page 703
- §24 [Converting tables to HTML](#) on page 727

For more information about HTML Help, see the Microsoft HTML Help home page, accessible through the Microsoft Library:

<http://msdn.microsoft.com/library>

9.1 Understanding how Mif2Go produces HTML Help

Microsoft HTML Help is specialized for use with Microsoft HTML Help Workshop, which is used to compile the HTML files **Mif2Go** generates.

Note: HTML Help does not always perform as documented; there are many defects, some pieces are missing, and the software is no longer maintained. These are issues that **Mif2Go** cannot address.

To produce HTML Help, **Mif2Go** does the following:

- creates an HTML Help project file
- generates HTML topic files from your FrameMaker document
- optionally runs HTML Help Workshop to compile the HTML Help project.

<i>Initial project file</i>	When you create a Mif2Go HTML Help project, Mif2Go automatically generates a starting Microsoft HTML Help project file. This file is named for your FrameMaker document, with extension <code>.hhp</code> , and placed in the project directory. For example, if you are converting <code>MyDoc.book</code> , Mif2Go creates an HTML Help project file named <code>MyDoc.hhp</code> .
<i>Project file can be regenerated</i>	The HTML Help project file contains the basic information needed to compile your HTML Help project. Once created, Mif2Go does not touch the HTML Help project file again, though you can tell Mif2Go to regenerate it each time you run a conversion; see §9.3.9 Regenerating the HTML Help project file on page 301.
<i>Project file can be edited</i>	You can edit the HTML Help project file yourself, either in a plain-text editor such as Notepad, or in HTML Help Workshop. Editing in HTML Help Workshop is risky, because the editing facility has known defects.
<i>Compiled CHM file is the final output</i>	The HTML Help project file and the generated HTML files become input for HTML Help Workshop, which compiles all the HTML topic files into a single “Compiled HTML File” named for your FrameMaker document, with extension <code>.chm</code> . The CHM file is the file you distribute, for use with the Microsoft HTML Help viewer. You can direct Mif2Go to run the compilation, or you can use HTML Workshop yourself to compile.
<i>CHM files must be “unblocked”</i>	Microsoft introduced “security” features that require each CHM file to be explicitly “unblocked”, something you will have to tell your users. To unblock a CHM, right-click its icon in Windows Explorer, select Properties , and click Unblock , then click Apply , then OK . After that, when you double-click the CHM, it opens fine. If you select Properties again, the area where the Unblock button was is now blank.
<i>View compiled file with the viewer</i>	The only way to access all features of HTML Help is to use the HTML Help viewer; you cannot view a CHM file with a Web browser. Neither Internet Explorer nor Firefox is able to display the tri-pane and search windows, although Internet Explorer (but not Firefox) can be persuaded to look inside a <code>.chm</code> on your local system. This is true regardless of the tool used to generate HTML Help.
<i>View uncompiled files with a browser</i>	If you are interested only in viewing HTML topic files, without the Contents, Index, Search, or the toolbar buttons, you can use a browser. If that is your intention, when you use Mif2Go to generate HTML Help, choose configuration options that do not produce <code><object></code> tags (which means no pop-ups or secondary windows); those tags appear in Firefox as extra spaces, and do not work as intended. You are better off converting to XHTML (or, if necessary, standard HTML); including in the conversion your FrameMaker TOC and IX to provide Contents and Index; and adding navigation (see §20 Providing navigation in HTML on page 627) to the top and bottom of each output page. Also see §9.6.2 Specifying href link syntax for HTML Help on page 308.

9.2 Understanding why Unicode is not the answer

Microsoft HTML Help does not use Unicode; instead it uses Windows code pages. This means that characters with glyphs that are not present in the default code page (for Western languages this is ANSI code page 1252) might not display correctly, and will interfere with use of TOC, index, and search functions

People often think they can get away with using Unicode encoding instead of code-page encoding, because the HTML Help viewer uses Internet Explorer to display the topic pane, and Internet Explorer does understand Unicode. However, if you use any non-ANSI (above `U+007F`) characters, search will not work right, and if any of your non-ANSI characters appear in titles or in index terms, the TOC and index will not work right, either. If you are processing a language with accented characters, such as German, you cannot get

away with Unicode in the topic pane. For example, Unicode represents code points from hexadecimal A0 to FF as two-byte UTF-8 sequences, and code page 1252 represents them as single characters. So even though the code points are the same, and the display looks fine, search fails because the single byte in the search string does not match the two bytes in the UTF-8 encoding.

With a few isolated symbols, you might get away with Unicode content, but it is not good practice. **Mif2Go** goes to considerable lengths to convert from Unicode to code page for HTML Help. It is not trivial; for Asian languages, **Mif2Go** uses enormous look-up tables and dozens of lines of C++ code. It is a Bad Idea to blow it off and use Unicode in any form (including numeric character references) instead.

It might be easy to dismiss all this when your language is English, but the rest of the world feels differently.

See also:

§9.13 [Generating HTML Help in non-Western languages](#) on page 331

§21.5 [Assigning properties to text formats](#) on page 653

9.3 Setting up an HTML Help project

To produce an HTML Help system you will need HTML Help Workshop, which you can download from the Microsoft Library:

<http://msdn.microsoft.com/en-us/library/ms669985.aspx>

Documentation for HTML Help Workshop is available from the same site.

If you plan to generate HTML Help in non-Western languages, you will also need the ICU library; see §9.13 [Generating HTML Help in non-Western languages](#) on page 331.

In this section:

§9.3.1 [Creating an HTML Help project](#) on page 297

§9.3.2 [Choosing set-up options for an MS HTML Help project](#) on page 298

§9.3.3 [Deciding where to locate configuration settings](#) on page 299

§9.3.4 [Organizing source files for HTML Help](#) on page 299

§9.3.5 [Specifying a project title for HTML Help](#) on page 300

§9.3.6 [Deciding whether to compile HTML Help](#) on page 300

§9.3.7 [Naming project and compiled files for HTML Help](#) on page 300

§9.3.8 [Specifying a starting topic file for HTML Help](#) on page 301

§9.3.9 [Regenerating the HTML Help project file](#) on page 301

§9.3.10 [Locating graphics files for HTML Help](#) on page 302

See also:

§7.2.1 [Checking automatic Help topic assignments](#) on page 203

9.3.1 Creating an HTML Help project

To create an HTML Help project:

1. Create a project directory for HTML files, separate from the directory where your FrameMaker document is located. Optionally, create a subdirectory for graphics files.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).

3. Name your project, and browse to the project directory you created in [Step 1](#).
4. Choose output type MS HTML Help and click **OK**.
5. Check options in the *Set Up MS HTML Help Project* dialog (see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425).
6. Use a text editor to edit the resulting `_m2htmlhelp.ini` configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).
7. **Important:** To make the configuration fit your usage of headings to start topics, pay special attention to sections `[HelpContentsLevels]` (see §7.2.1 [Checking automatic Help topic assignments](#) on page 203) and `[HTMLParaStyles]` (see §18.2 [Splitting files](#) on page 586).
8. **Important:** All the files you include in a compiled HTML Help system must be located in or below the directory that contains the `.hhp` project file. See §9.3.3 [Deciding where to locate configuration settings](#) on page 299.

9.3.2 Choosing set-up options for an MS HTML Help project

When you select MS HTML Help as the output type for a new project, the *Set Up* dialog shown in [Figure 9-1](#) opens. [Table 9-1](#) shows the corresponding settings in the configuration file. *You must edit the configuration file to specify additional options.*

See also:

§3.4 [Choosing project set-up options](#) on page 79

§7 [Producing on-line Help](#) on page 199

Figure 9-1 Set Up MS HTML Help Project

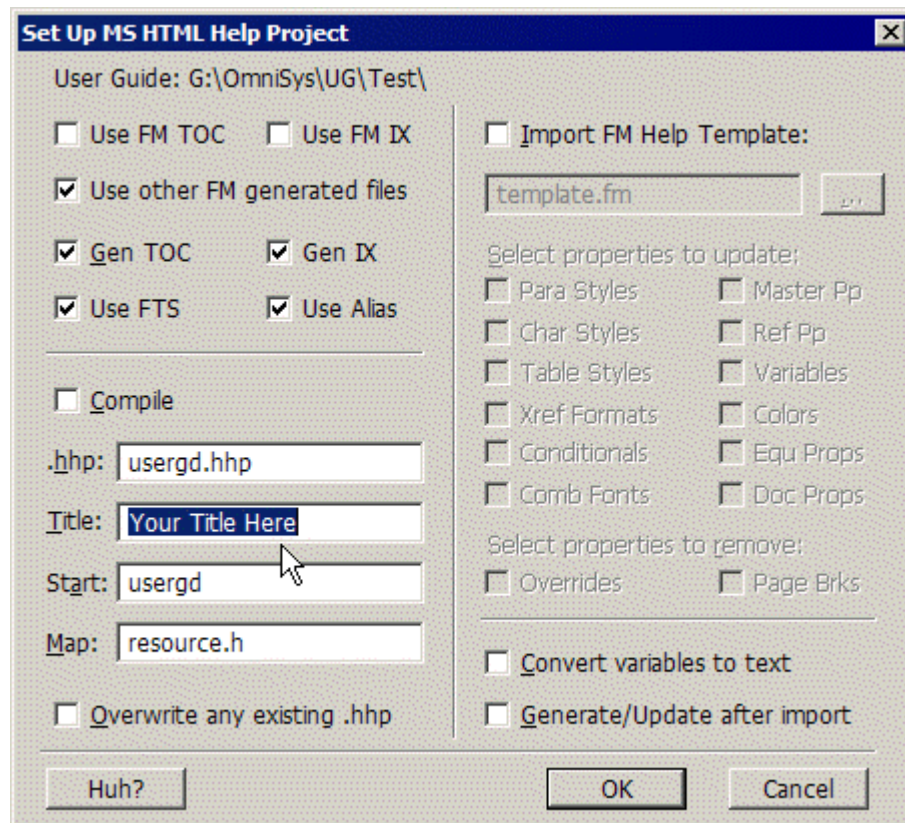


Table 9-1 HTML Help set-up options and configuration settings

Set-up Option	Configuration file Section	Setting	Default	Ref.
Gen TOC	[MSHtmHelpOptions]	ListType=Contents Both RefFileType=Body Full	Both <i>Depends*</i>	7.3.4.1 9.9.1
Gen IX	[MSHtmHelpOptions]	ListType=Index Both RefFileType=Body Full	Both <i>Depends*</i>	7.3.4.1 9.9.1
Use FTS	[MSHtmHelpOptions]	UseFTS=Yes	Yes	9.11
Use Alias	[MSHtmHelpOptions]	MakeAliasFile=Yes	Yes	9.12.1.1
Compile:	[Automation]	CompileHelp=Yes	No	9.14
.hhp:	[MSHtmHelpOptions]	HHPFileName= <i>MyDoc</i> .hhp	<i>MyDoc</i> .hhp	9.3.7
	[MSHtmHelpOptions]	DefaultChmFile= <i>MyDoc</i>	<i>MyDoc</i>	9.3.7
Title:	[MSHtmHelpOptions]	HelpFileTitle= <i>My Title</i>	Your Title Here	9.3.5
Start:	[MSHtmHelpOptions]	DefaultTopicFile= <i>Filename</i>	<i>MyDoc</i>	9.3.5
Map:	[MSHtmHelpOptions]	CshMapFile= <i>resource.h</i>	<i>resource.h</i>	9.12.1.1
Overwrite .hhp	[MSHtmHelpOptions]	WriteHelpProjectFile=Yes	No	9.3.9
* The default is Body for FrameMaker books, Full for single document files.				

9.3.3 Deciding where to locate configuration settings

When you set up an HTML Help project from within FrameMaker, if configuration file `_m2htmlhelp.ini` is not already present in the project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

*Which
configuration file?*

To configure HTML Help output, add settings to one of the following files, depending on the desired scope of each setting:

Scope	Configuration file	Location
Current project only	<code>_m2htmlhelp.ini</code>	Current project directory
All HTML Help projects	<code>local_m2htmlhelp_config.ini</code>	<code>%omsyshome%\m2g\local\config\</code>

See §30.5 [Deciding which configuration file to edit](#) on page 856.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §13 [Converting to HTML/XHTML](#) on page 423
- §18 [Splitting and extracting files](#) on page 585
- §21 [Mapping text formats to HTML/XML](#) on page 645
- §23 [Including graphics in HTML](#) on page 703
- §24 [Converting tables to HTML](#) on page 727

9.3.4 Organizing source files for HTML Help

Compiled HTML Help has the following limitation on file placement: a CHM can contain files located only in the same directory as the `.hhp` file (HTML Help project file) or in a subdirectory. Sibling directories, parent directories, and absolute paths elsewhere do not work. CHM content is organized in an internal file system that duplicates the external file structure, but with the directory containing the `.hhp` file as the root. Therefore, references to directories outside this structure do not work.

If your project includes FrameMaker files on different paths that reference each other, you might have to reorganize them to conform to this HTML Help limitation. If your project includes referenced graphics files, **Mif2Go** can copy the graphics files to the project directory (or a subdirectory) before beginning the conversion. See §9.3.10 [Locating graphics files for HTML Help](#) on page 302.

9.3.5 Specifying a project title for HTML Help

The title of your HTML Help project appears in the title bar of the HTML Help viewer. When you set up a new HTML Help project, you can specify a title in the *Set Up* dialog; see §9.3.2 [Choosing set-up options for an MS HTML Help project](#) on page 298. You can also specify a title in the configuration file.

To specify a title for your help project:

```
[MShtmlHelpOptions]
; HelpFileName = title to put in project file;
; default is filename or bookname
HelpFileName=Title of my project
```

If your CHM file will be used in locales other than US English, also see §9.13 [Generating HTML Help in non-Western languages](#) on page 331.

9.3.6 Deciding whether to compile HTML Help

FrameMaker version 8 and above use Unicode. The HTML Help compiler does not support Unicode, and instead uses code-page mappings. For compiled HTML Help, **Mif2Go** maps Unicode characters to the correct code page.

If what you need is uncompiled HTML Help files in Unicode, possibly including contents, index, and search files, you can direct **Mif2Go** *not* to compile HTML Help:

```
[Automation]
CompileHelp = No
```

To omit code-page mapping when you are not going to compile HTML Help:

```
[MShtmlHelpOptions]
; UseCodePage = Yes (default, required for CHM compile), or No
; (for use in further processing where other encodings are OK)
UseCodePage = No
```

To produce Japanese, Chinese, or Korean code-page output, such as for HTML Help in Japanese, you need ICU DLLs: `icudt40.dll` (13MB) and `icuuc40.dll` (1MB). These DLLs are available in archive `icu401.zip` (6 MB), which you can download from the Omni Systems Web site. See:

§1.1.4 [Languages and character sets](#) on page 53

§9.13 [Generating HTML Help in non-Western languages](#) on page 331

§9.14 [Compiling and testing HTML Help](#) on page 333.

9.3.7 Naming project and compiled files for HTML Help

By default, **Mif2Go** uses the name of your FrameMaker book or document file for both the project file (`.hhp`) and the compiled file (`.chm`). When you set up a new HTML Help project, you can specify a different name in the *Set Up* dialog; see §9.3.2 [Choosing set-up options for an MS HTML Help project](#) on page 298. You can also specify different file names in the configuration file; and you must do so for the CHM file, if you are reusing an existing configuration file for a new project.

Help project file To specify the .hhp file name:

```
[MSHtmlHelpOptions]
; HHPFileName = MS HTML Help project file used for compilation
HHPFileName = myproj.hhp
```

The default value is the name of your FrameMaker book or document file.

Note: Neither the name of the file nor the path to the file may contain spaces. Do not enclose the name in quotes.

Compiled file To specify the CHM file name:

```
[MSHtmlHelpOptions]
; DefaultChmFile = name of .chm for project if not in [ChmFiles]
DefaultChmFile = myproj
```

If your project includes links to other HTML Help projects, also see §9.15 [Mapping and merging CHM files](#) on page 336.

9.3.8 Specifying a starting topic file for HTML Help

By default, **Mif2Go** puts the name of your FrameMaker document in the .hhp as the name of the first page. If the first page really should be the first *split* file (see §18 [Splitting and extracting files](#) on page 585), you must edit the .hhp (either in Notepad or in HTML Help Workshop) to insert the actual name of the first-page file (which might be something like aal23456.htm):

```
[OPTIONS]
Default topic=realfilename.htm
```

Also specify the starting topic in configuration file m2htmlhelp.ini:

```
[MSHtmlHelpOptions]
; DefaultTopicFile = starting topic file name (no extension)
DefaultTopicFile=realfilename
```

Then if your .hhp is rewritten, it will have the correct value.

After compiling your project, if you open the .chm in HTML Help and the first page produces an error such as “This page cannot be displayed”, you might have to edit the .hhp to specify the correct name for the first-page file.

9.3.9 Regenerating the HTML Help project file

When you use **Mif2Go** to generate HTML Help from within FrameMaker, **Mif2Go** writes an .hhp project file during set-up, and rewrites it later only under certain conditions.

To specify whether **Mif2Go** should generate the .hhp project file anew each time you run the conversion:

```
[MSHtmlHelpOptions]
; WriteHelpProjectFile = Yes (write each time) or No; if no setting,
; write only if the file does not already exist.
WriteHelpProjectFile = Yes
```

The values you can specify for WriteHelpProjectFile have the following effects:

- | | |
|--------|--|
| Yes | If the .hhp file is present, Mif2Go overwrites it. |
| No | Mif2Go does not overwrite the .hhp file. |
| (none) | If the configuration chain contains no WriteHelpProjectFile setting at all, Mif2Go writes an .hhp file, but <i>only if the .hhp file is not already present</i> . |

Mif2Go closes the .hhp file after writing it; so, if you had the .hhp file open in HTML Help Workshop when **Mif2Go** rewrote it, you could get an access violation. If you were using Notepad to edit the .hhp file, on save Notepad would just write the old file over the rewritten one.

If you use HTML Help Workshop to make changes that are not reflected in the configuration file, and they are changes you want to keep, you can prevent **Mif2Go** from overwriting them by setting `WriteHelpProjectFile=No`.

If you set `WriteHelpProjectFile=Yes` and then later decide to modify the .hhp file directly, be sure to set `WriteHelpProjectFile=No`; otherwise your edits will be wiped out the next time you run the conversion.

If the changes you make via HTML Help Workshop are limited to defining windows, you can add those definitions to your **Mif2Go** configuration file to preserve them; see §9.8.1 [Defining secondary windows for HTML Help](#) on page 317.

9.3.10 Locating graphics files for HTML Help

A .chm file can include only files that are located in the same directory as the .hhp file, or in a subdirectory of that directory. If your graphics files are located elsewhere, they must be copied to the .hhp directory or subdirectory. **Mif2Go** can do this for you.

To tell **Mif2Go** to fetch your referenced graphics:

```
[Automation]
WrapAndShip = Yes
CopyOriginalGraphics = Yes
```

When `CopyOriginalGraphics=Yes`, **Mif2Go** follows the file paths in your FrameMaker source to find the graphics files to copy.

To tell **Mif2Go** where to put copies of the graphics (for example):

```
[Graphics]
GraphPath = ./graphics
```

The path you specify for `GraphPath` should be relative to the wrap directory (see §35.3 [Understanding path values for deliverables](#) on page 957). This path will be used in HTML output, as the relative path from the HTML files to their referenced graphics. If you use backslashes in the path, **Mif2Go** converts them to forward slashes before inserting the references in your HTML output. If you specify `CopyOriginalGraphics=Yes`, **Mif2Go** copies graphics files to the directory specified by `GraphPath`, after generating HTML files.

See also:

§9.14 [Compiling and testing HTML Help](#) on page 333

§23.3 [Locating graphics files for HTML](#) on page 704

§35.7 [Placing graphics files for distribution](#) on page 965

9.4 Customizing HTML Help display features

In this section:

§9.4.1 [Using CSS and font tags with HTML Help](#) on page 303

§9.4.2 [Eliminating graphic and table indents from HTML Help](#) on page 303

§9.4.3 [Adding tabs and toolbar buttons to HTML Help](#) on page 303

§9.4.4 [Adding expandable sections to HTML Help](#) on page 305

9.4.1 Using CSS and font tags with HTML Help

The Microsoft HTML Help viewer, which is based on Internet Explorer, supports CSS (cascading style sheets); however, not every CSS feature is supported. Although CSS is probably the best way to set display properties for HTML Help, you might have to experiment. If a CSS feature you try does not seem to work, check the HTML Help Workshop on-line Help, or check with Microsoft to make sure HTML Help supports that feature.

For on-demand font resizing via the **Font** button (see §9.4.3 [Adding tabs and toolbar buttons to HTML Help](#) on page 303), CSS font sizes must be in relative units: em, ex, or %. For best practice, use em. By default, **Mif2Go** generates CSS entries using absolute pt units for font size and line height. To change the units, see §22.8.3 [Specifying CSS size values and units of measurement](#) on page 699.

If you are *not* using CSS, by default **Mif2Go** uses tags for HTML Help. That is, when UseCSS=No, you have to turn tags off if you do not want them:

```
[HTMLOptions]
NoFonts=Yes
```

See also:

§21.7.4 [Including or excluding font tags](#) on page 665

§22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684

9.4.2 Eliminating graphic and table indents from HTML Help

For HTML Help, with its narrow windows, you might want to eliminate all graphic and table indents. Specify the following settings:

```
[GraphIndents]
*=0

[TableIndents]
*=0
```

Also, you might want to edit your .css file to eliminate most left indents in text.

9.4.3 Adding tabs and toolbar buttons to HTML Help

You can use HTML Help Workshop to enable additional tabs and toolbar buttons for navigation and other features. For example, rather than create links for **Prev** and **Next**, you can enable built-in browse buttons for this purpose.

Note: Enabling browse buttons requires a binary TOC, which can cause problems with mid-topic TOC links; see §9.9.6 [Providing mid-topic contents links in HTML Help](#) on page 323. Also, a binary TOC is not compatible with merged CHM files; see §9.15.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 339.

To enable additional HTML Help tabs and toolbar buttons:

1. Set the following option in your project configuration file, to avoid overwriting the changes you are about to make to the HTML Help .hhp project file:

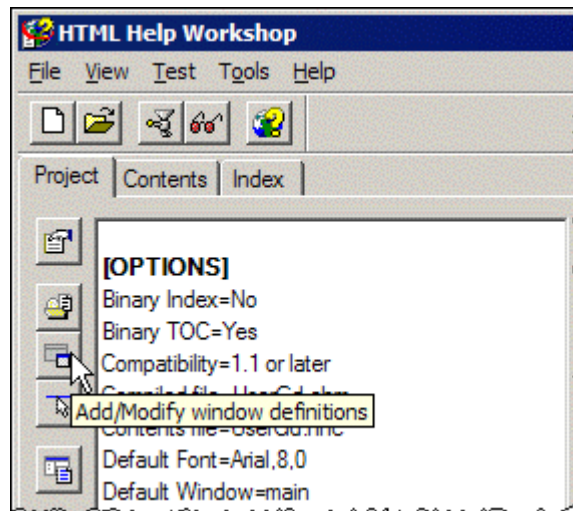
```
[MShtmlHelpOptions]
WriteHelpProjectFile=No
```

See §9.3.9 [Regenerating the HTML Help project file](#) on page 301

2. In HTML Help Workshop, click **File > Open**.

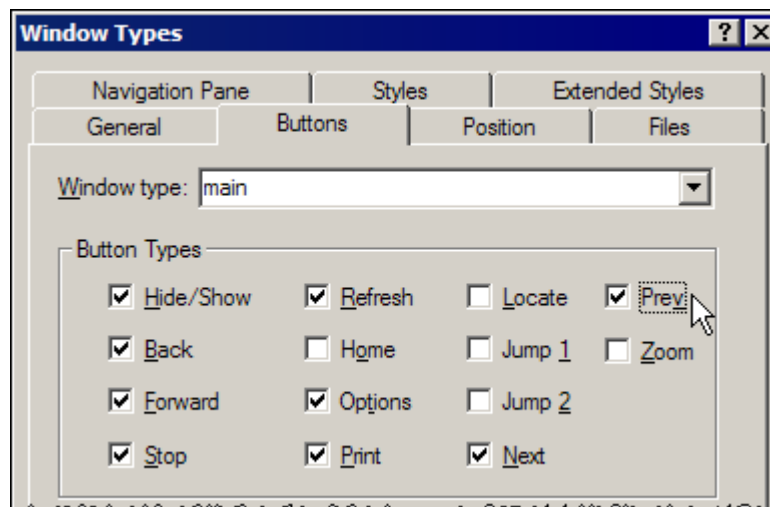
3. In the *Open* dialog, type:
I 'm MSDN
(straight single quote, not curly) and click **Open**.
4. Select your .hhp project file.

Figure 9-2 HTML Help Workshop Project tab



5. On the **Project** tab toolbar, click **Add/Modify window definitions**, as shown in Figure 9-2.
6. If the *Add a New Window Type* dialog opens, type **main**, and click **OK**; the *Window Types* dialog opens, as shown in Figure 9-3 on page 304.
7. For window type choose (or specify) **main**.
8. To add a favorites tab to your project, select the **Navigation Pane** tab and check **Favorites tab**.
9. To add toolbar buttons, select the **Buttons** tab.

Figure 9-3 HTML Help Workshop Window Types



10. Check the button types you want HTML Help to display in the toolbar. For example:
 - to add **Previous** and **Next** buttons for browsing, check **Prev** and **Next**
 - to add a **Font** button for text resizing, check **Zoom**.

Note: If you check **Zoom**, also specify relative units for font sizes in CSS; see §9.4.1 [Using CSS and font tags with HTML Help](#) on page 303.

11. Click **OK**.
12. Click **File > Save Project**.

*Browse buttons
require a binary
TOC*

To use **Prev** and **Next** browse buttons in your help file, you must also compile a binary table of contents. Under [Options] in the .hhp file, add the following line:

```
Binary TOC=Yes
```

Or, you can set this option in HTML Help Workshop:

1. Select the **Project** tab.
2. On the **Project** tab toolbar, click **Change project options**; the *Options* dialog opens.
3. Select the **Compiler** tab.
4. Check **Create a binary TOC**.
5. Click **OK**.
6. Click **File > Save Project**.

9.4.4 Adding expandable sections to HTML Help

The HTML Help Workshop help file explains how to add the required HTML and JavaScript code for expanding sections. With **Mif2Go** macros you can automate insertion of this code in the output.

For example, suppose you want to use a link:

Click for more

which, when clicked, displays additional text:

Here is a more detailed explanation.

You could apply a special paragraph format to the link text (for example, *ClickLink*) and another format to the expandable text (for example, *ClickText*). In your configuration file you would include macros that contain the required HTML code for each paragraph type. If the link text was always the same, you could include that in the macro too, and just have the text for the expanded part in FrameMaker. Or, you could use a button (see §9.7.6 [Creating buttons for other types of related-topic links](#) on page 317).

9.5 Creating pop-ups for HTML Help

HTML Help supports only text pop-ups. By itself, HTML Help does not allow *any* font changes in pop-ups, not even bold or italics; nor any images; nor any HTML code. To include these features in an HTML Help pop-up, you need a third-party plug-in or program, such as WinHelp.


In this section:

- §9.5.1 [Using HTML Help for pop-ups](#) on page 306
- §9.5.2 [Using KeyHelp for pop-ups](#) on page 306
- §9.5.3 [Using WinHelp for pop-ups](#) on page 307

9.5.1 Using HTML Help for pop-ups

Although you cannot specify font or text property changes *within* an HTML Help pop-up, you can specify text attributes for the entire pop-up. The following settings apply to *all* pop-ups. For example:

```
[MShtmlHelpOptions]
; MS HTML Help Popup options, for use with Hypertext Alert markers
; PopFont = Facename[,point size[,charset[,color
;   [,PLAIN BOLD ITALIC UNDERLINE]]]]
PopFont=Helvetica,10,,PLAIN
; PopMargins = left margin, right margin (in pixels)
PopMargins=9,9
; PopColors = foreground, background (in decimal RGB, -1 is default)
PopColors=-1,-1
```

HTML Help text pop-ups are limited to 243 characters in length. You put the pop-up text itself inside a FrameMaker hypertext **alert** marker (see §34.1.2 [Using markers to add links and instructions](#) on page 935), and indicate the span of the hotspot with a character format that includes the marker. If you do not use a character format, the entire paragraph becomes the hotspot. If you are viewing this text in HTML Help,  *is an example.*

You can put pop-ups in image maps. In FrameMaker, an information alert box containing the pop-up text appears when you **Ctrl+Alt**-click the hotspot.

9.5.2 Using KeyHelp for pop-ups

KeyHelp is a freeware DLL that is part of Ralph Walden's Key Tools. KeyHelp allows you to embed better pop-ups in HTML Help. For information about Key Tools, see:

http://grainge.org/pages/authoring/reverse_engineering/reverse_engineering.htm

For KeyHelp pop-ups to work, the KeyHelp ActiveX control, `keyhelp.ocx`, must be installed and registered on each user's system.

You must use **Mif2Go** macros (see §28 [Working with macros](#) on page 787) to construct the HTML code that is needed around the content. For example, in your configuration file, you could include the following settings:

```
[Inserts]
Head=<$KeyPopup>

[KeyPopup]
<script language="JavaScript" type="text/javascript">
var KeyPopup;
function KeyDisplayPopup(URL) {
  if (!KeyPopup){
    KeyPopup = new ActiveXObject("KeyHelp.KeyPopup");
  }
  KeyPopup.DisplayURL(URL,-1,-1);
}
</script >
```

The cross-reference format you use to reference the pop-up should apply a character format; for example, *PopText*:

```
<PopText><$paratext></>
```

To make the cross references call the KeyHelp DLL, include the following settings:

```
[XrefStyles]
PopText=LinkSrc

[XrefStyleLinkSrc]
PopText=JavaScript:KeyDisplayPopup('myproj.chm:<$$_linksrc')'
```

Make sure the material to be popped up is in a file of its own. For example, if you are using glossary entries as pop-up topics:

```
[HtmlStyles]
GlossaryTerm=Split Title
```

9.5.3 Using WinHelp for pop-ups

This is the method Microsoft uses for Office 2000. WinHelp works best when pop-ups are called from the application, rather than from other topics in the HTML Help file. See §8.9 [Creating jumps and pop-ups for WinHelp](#) on page 272.

9.6 Creating links and hypertext jumps in HTML Help

In this section:

§9.6.1 [Creating hypertext jumps to other CHM files](#) on page 307

§9.6.2 [Specifying href link syntax for HTML Help](#) on page 308

§9.6.3 [Linking to external files from compiled HTML Help](#) on page 308

See also:

§9.7 [Creating related-topic links for HTML Help](#) on page 309

9.6.1 Creating hypertext jumps to other CHM files

If you are using **newlinks** and **gotolinks** to link to HTML files created from a different FrameMaker document and compiled into a different .chm, you must specify how the CHM files are mapped; see §9.15.1 [Interlinking multiple CHM files](#) on page 336. Then **Mif2Go** can generate the proper jump reference for you.

Opening topic of first file

To jump to the opening topic of another CHM file, the simplest method is to insert in your document a FrameMaker hypertext marker with the following marker content:

```
message URL someother.chm
```

If *someother.chm* is not registered in the Windows registry, also include the path, possibly relative. (For commercial use, it is best to register .chms in the Windows registry during installation.)

Opening topic of any file

To jump to a file that is within *someother.chm*, add the name of the target file to the marker content; for example, to get to the “a” anchor inside *letters.htm* in *someother.chm*:

```
message URL someother.chm::/letters.htm#a
```

Specific topic

To jump to a specific topic in *someother.chm*, when the topic comes from a FrameMaker file that is being split (so you do not know the .htm file name ahead of time), insert a regular FrameMaker cross reference. Run the **Mif2Go** conversion to *someother.chm* first, then copy the .ref file for *someother.chm* into your conversion project directory, and add this setting to the configuration file:

```
[ChmFiles]
letters=someother
```

For links to non-CHM files, see §9.6.3 [Linking to external files from compiled HTML Help](#) on page 308.

9.6.2 Specifying href link syntax for HTML Help

Links in HTML Help can be of two forms: generic HTML href links for jumps within a single CHM file, or href links with a special syntax that includes the target .chm name for jumps to locations in other CHM files. If you are using framesets in HTML Help, target content might be displayed differently for these two forms of links.

By default, **Mif2Go** uses both forms for HTML Help:

- generic HTML links for jumps within the CHM file you are generating
- special syntax for jumps to other CHM files.

For links to non-CHM files, see §9.6.3 [Linking to external files from compiled HTML Help](#) on page 308.

If your configuration file lists other CHM files in section [ChmFiles], jumps to those files use the special syntax that includes the .chm name. Unless you specify otherwise, jumps within the default CHM file are of the generic form. (If you are using multiple CHM files, this file is the DefaultChmFile listed in section [MShtmlHelpOptions]; see §9.15.1 [Interlinking multiple CHM files](#) on page 336.)

Force all links to use special syntax

To force *all* links to use the special syntax that includes the .chm name:

```
[MShtmlHelpOptions]
; UseChmInLinks = No (default, for same .chm or for uncompiled help,
;   where normal links are needed)
;   or Yes (always use ChmFormat at start of links)
UseChmInLinks=Yes
```

Use generic form for single .chm, uncompiled Help

When UseChmInLinks=No (the default), links to destinations within the default CHM file are of the generic HTML href form; use this setting to produce either of the following:

- a single CHM file that does not include href links to other CHM files
- uncompiled HTML Help that has no CHM file, and that works in a Web browser.

Use special syntax for links to other CHM files

When UseChmInLinks=Yes, by default *all* links are of the form:

```
<a href="mk:@MSITStore:Helpfile.chm::/topicfile.htm#anchor">
```

For example:

```
<a href="mk:@MSITStore:ugmif2go.chm::/z12x1391027.htm#Rz12x18218">
```

Specify format for special syntax

To specify a format for the start of the link:

```
[MShtmlHelpOptions]
; ChmFormat = format to use when UseChmInLinks is set, where
;   the first %s is the chm name and the second %s is the filename
ChmFormat=mk:@MSITStore:%s.chm::/%s
```

For example, if all users of your compiled HTML Help system will be running Internet Explorer 4.0 or a later version, you can direct **Mif2Go** to use the following form for all links instead of the default special syntax:

```
[MShtmlHelpOptions]
UseChmInLinks=Yes
ChmFormat=ms-its:%s.chm::/%s
```

See HTML Help Workshop for more information.

9.6.3 Linking to external files from compiled HTML Help

Compiled HTML Help allows calls to a non-CHM file only if the path to that file is absolute. If your Help system is always installed to the same drive and path, you can hardcode the path in the link, but that is not usually the case. Instead, you can use

JavaScript to determine the location of the calling CHM file at run time and prefix that path (possibly modified with additional elements) to the target file name in the link. In effect, this method provides a relative path. See the following MSDN article for details:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/vsconocxscriptslinkchm.asp>

The MSDN article says the external file must be in the same directory as the calling CHM file, but that is not true. You just have to know where the external file is *relative to* the calling CHM file, and use the external file name with that relative path; the JavaScript adds the first part of the path. You can use a **Mif2Go** macro to provide the JavaScript; see §28 [Working with macros](#) on page 787.

Linking to Web sites

For external links on the Web, you can use a hypertext **message URL** marker with the full Web link. However, you might encounter security issues in Internet Explorer, so test thoroughly on machines with the latest security patches. You might have to tweak Windows Registry settings, which you cannot do from within HTML Help. For example, on Windows 2000:

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\HTMLHelp\1.x\HHRestrictions]
"MaxAllowedZone"=dword:00000004
"EnableFrameNavigationInSafeMode"=dword:00000001

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\HTMLHelp\1.x\ItssRestrictions]
"MaxAllowedZone"=dword:00000004
```

9.7 Creating related-topic links for HTML Help

You can use ALinks and KLinks in compiled HTML Help. ALinks also work in uncompiled HTML Help, provided you use the HTML Help viewer.

In this section:

- §9.7.1 [Adding ALink keywords for HTML Help](#) on page 309
- §9.7.2 [Adding ALink and KLink jumps for HTML Help](#) on page 309
- §9.7.3 [Configuring ALink and KLink jumps for HTML Help](#) on page 310
- §9.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 312
- §9.7.5 [Using the same format or marker for ALink keywords and jumps](#) on page 312
- §9.7.6 [Creating buttons for other types of related-topic links](#) on page 317

9.7.1 Adding ALink keywords for HTML Help

You can insert ALink keywords for HTML Help either with markers or with paragraph formats; see §7.6.4 [Adding related-topic link keywords in FrameMaker](#) on page 221. ALink keywords should be single terms; use spaces or other punctuation in ALink keywords at your own risk.

9.7.2 Adding ALink and KLink jumps for HTML Help

You can use markers or paragraph formats for ALink and KLink jumps; however, **Mif2Go** provides support only for markers. In HTML Help, a related-topic jump is implemented with an <object> linked to the HTML Help OLE control:

Markers If you use markers for ALink or KLink jumps, **Mif2Go** provides the <object> macro code automatically; see §9.7.3 [Configuring ALink and KLink jumps for HTML Help](#) on page 310. Put the markers in paragraphs by themselves, located where you want the jump to appear in output.

Formats If you use paragraph formats for ALink or KLink jumps, you have to provide macro code for the ALink object; see §9.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 312.

See also §7.6.5 [Adding ALink and KLink jumps in FrameMaker](#) on page 222.

Note: Because both ALink/KLink jumps and secondary windows/pop-ups use objects in HTML Help, they cannot be combined; for example, you cannot display an ALink-accessed page in a secondary window.

ALinks work in uncompiled HTML Help, if you use the HTML Help viewer instead of a Web browser.

9.7.3 Configuring ALink and KLink jumps for HTML Help

When you use **Go to URL** markers for ALink or KLink jumps (see §7.6.5 [Adding ALink and KLink jumps in FrameMaker](#) on page 222), you can specify values in the configuration file for several properties of the resulting <object>s that **Mif2Go** creates:

```
[MShtmlHelpOptions]
;LinkType = Button (default), Chiclet, Graphic, Icon, Shortcut, Text
LinkType=Button
;LinkFlags = "1" (show dialog even for one item),
; " ,1" (if no items, make button disappear), or
; empty (if only one item, take the jump directly)
LinkFlags=1
;LinkEmptyTopic = name of .htm topic file to show if no items match
; at all; otherwise, unless LinkFlags= ,1, the Not Found complaint
; will be used.
;LinkEmptyTopic=noitems.htm
;LinkButtonWidth = pixels, if LinkType = Button, Graphic, or Icon
LinkButtonWidth=100
;LinkButtonHeight = pixels, if LinkType = Button, Graphic, or Icon
LinkButtonHeight=100
;LinkButtonText = Text: plus name on button, if LinkType=Button
LinkButtonText=Text:ALink
;LinkButtonGraphic = Bitmap: plus name of .bmp (only),
; if LinkType=Graphic
LinkButtonGraphic=Bitmap:mybutton.bmp
;LinkButtonIcon = Icon: plus name of .ico (only), if LinkType=Icon
LinkButtonIcon=Icon:mybutton.ico
;LinkTextFont = same syntax as PopFont above, if LinkType=Text
LinkTextFont=Helvetica,10,,PLAIN
; LinkText = Text: plus text to use for link, if LinkType=Text
LinkText=Text:Related Topics
```

Table 9-2 on page 311 shows the properties you can configure, the values you can specify for each property, and the effect of each value.

<i>Prefix the keyword</i>	<p>The base keyword for each property starts with Link; for example, LinkFlags. You must add a prefix to the base keyword to create a valid setting:</p> <ul style="list-style-type: none"> • To specify an ALink property, prefix the base name with A; for example, ALinkFlags= ,1. • To specify a KLink property, prefix the base name with K; for example, KLinkButtonWidth=50.
<i>All jumps</i>	Properties you specify this way apply to all ALink or KLink jumps in your document.
<i>Selected jumps</i>	To configure properties for an individual ALink or KLink jump, insert a Link* marker just before the jump marker. The Link* marker-type name is the same as the name of the property, and the marker content is the value you want to assign to that property. For

example, to provide a different label for one particular text-style ALink jump, in your FrameMaker document, just before the **ALink** jump marker, insert a **LinkText** marker whose content is the alternate label.




Specify all relevant properties

For reasonable-looking output you should specify values for all properties that apply, because the default is to include only a few properties in the <object>. For example, for an ALink jump the default object **Mif2Go** generates looks like the following:

```
<object id="hhctrl1" type="application/x-oleobject"
  classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
  width="100" height="100">
<param name="Command" value="ALink">
<param name="Button" value="Text:ALink">
<param name="Flags" value="1">
<param name="Item1" value="">
<param name="Item2" value="ALink keyword">
</object>
```

This gets you a button labeled **ALink**, probably not what you wanted.

Table 9-2 ALink and KLink jump properties for HTML Help

Property keyword*	Value	Effect
LinkType	Button <i>(default)</i>	Link is a button with LinkButtonWidth, LinkButtonHeight, and LinkButtonText attributes: 
	Chiclet	Link is a small button with no label: 
	Graphic	Link is a bitmap image, specified by LinkButtonGraphic
	Icon	Link is an icon, specified by LinkButtonIcon
	Shortcut	Link is a button with a shortcut icon: 
	Text	Link is text, with LinkText and LinkTextFont attributes
LinkFlags	1 <i>(default)</i>	Show the dialog even if only one target is found
	,,1	Omit the link if no targets are found
	<i>(none)</i>	Omit the dialog and jump directly if only one target is found; show "Not Found" if no targets are found
	1,,1	<i>Untested</i> ; should show the dialog for a solitary target, omit the link if no targets are found.
LinkEmptyTopic	noitems.htm	Name of .htm topic file to show if no targets are found
	<i>(none)</i>	Issue a "Not Found" complaint, unless LinkFlags=,,1
LinkButtonWidth	100 <i>(default)</i>	Width of image in pixels, when LinkType is Button, Graphic, or Icon
LinkButtonHeight	100 <i>(default)</i>	Height of image in pixels, when LinkType is Button, Graphic, or Icon
LinkButtonText	Text:ALink <i>(default for ALinks)</i>	Text: followed by a label for the button, when LinkType=Button
LinkButtonGraphic	Bitmap:mybutton.bmp	Bitmap: followed by the file name of the graphic, when LinkType=Graphic; must be .bmp

* The property keyword must be prefixed with "A" for an ALink property, or "K" for a KLink property.

Table 9-2 ALink and KLink jump properties for HTML Help (continued)

Property keyword*	Value	Effect
LinkButtonIcon	Icon:myicon.ico	Icon: followed by the file name of the icon, when LinkType=Icon; must have extension .ico
LinkText	Related Topics	Text: followed by text to use for the link when LinkType=Text
LinkTextFont	Helvetica,10,,PLAIN	Font to use when LinkType=Text; syntax is the same as for PopFont (see §9.5 Creating pop-ups for HTML Help on page 305)

* The property keyword must be prefixed with "A" for an ALink property, or "K" for a KLink property.

9.7.4 Rolling your own macros for ALink jumps in HTML Help

To use a paragraph format (for example, *ALinkJump*) for ALink jumps, you can assign property `CodeReplace` to the format to replace the paragraph with a button for the ALink jump:

```
[HTMLParaStyles]
ALinkJump=CodeReplace

[HtmlParaStyleCodeReplace]
ALinkJump=<$ALinkButton>

[ALinkButton]
<object id="hhctrl" type="application/x-oleobject"
  classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
  width="100" height="100">
  <param name="Command" value="ALink">
  <param name="Button" value="Text:Related topics">
  <param name="Flags" value="1">
  <param name="Item1" value="">
  <param name="Item2" value="first ALink keyword">
  . . .
  <param name="ItemN" value="last ALink keyword">
</object>
```

"Item1" in the macro definition specifies the path to the CHM file that contains the target topic(s); an empty value means the current CHM file.

"Item2" through "ItemN" in the macro definition each specify the value of an ALink keyword; see **HTML Help Workshop Help** for more information. You would have to edit the macro for each different ALink keyword.

However, instead of dedicating a special paragraph format to ALink jumps, you can provide additional macros to produce ALink jumps from the paragraph format (or marker type) you use for ALink keywords; see §9.7.5 [Using the same format or marker for ALink keywords and jumps](#) on page 312

9.7.5 Using the same format or marker for ALink keywords and jumps

You can use the same paragraph format, or the same marker type, both for ALink keywords and to produce a button for an ALink jump in HTML Help. Use configuration settings and **Mif2Go** macros to capture content and create a list of keywords for a topic, then use additional macros to build an ALink button for the topic.

With this technique you can even include multiple keywords in a single paragraph or marker. The only restriction is that ALink keyword paragraphs or markers must precede the location in each topic where you want the ALink jump to appear.

In this section:

- §9.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 313
- §9.7.5.2 [Creating a list of ALink keywords from markers](#) on page 314
- §9.7.5.3 [Initializing the ALink keyword list counter](#) on page 315
- §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315
- §9.7.5.5 [Positioning the ALink button in each HTML Help topic](#) on page 316
- §9.7.5.6 [Including multiple ALink keywords in a paragraph or marker](#) on page 316

9.7.5.1 Creating a list of ALink keywords from paragraphs

Suppose you use paragraph format *ALinkTarget* for ALink keywords. To capture keywords from *ALinkTarget* paragraphs, assign properties to extract the paragraph content:

```
[HTMLParaStyles]
ALinkTarget=ALink Raw CodeStore CodeAfter
```

ALink property The ALink property specifies that the content of each *ALinkTarget* paragraph is to be used for the ALink Name property of an HTML Help ALink object (*not* the button object, which you will construct with macros); see §7.6.4.2 [Adding related-topic keywords via format properties](#) on page 222.

Raw property The Raw property suppresses any HTML tags that would otherwise be generated; see §21.3.6 [Stripping paragraph properties](#) on page 650.

CodeStore property The CodeStore property causes the content of the *ALinkTarget* paragraph to be stored in macro variable \$\$ALinkTarget. (The value of a macro variable that has the same name as a paragraph format is the content of the current paragraph in that format; see §28.3.1 [Creating and invoking macro variables](#) on page 796.) The CodeStore property also removes the paragraph from text output; see §28.3.7.2 [Inserting code with the CodeStore property](#) on page 804.

CodeAfter property The CodeAfter property provides the means to do something further with macro variable \$\$ALinkTarget, which now contains an ALink keyword, plucked from the *ALinkTarget* paragraph:

```
[ParaStyleCodeAfter]
ALinkTarget=<$$Nkeys++><$$ALinkKeys[$$Nkeys]=$$ALinkTarget>
```

Store paragraph content in a list variable The [ParaStyleCodeAfter] code does the following:

- Increments a counter, Nkeys (which will be initialized to zero before each topic).
- Uses Nkeys to index a list variable, \$\$ALinkKeys (see §28.4 [Using multiple-value list variables](#) on page 806).
- Stores the content of macro variable \$\$ALinkTarget in the Nkeys slot in list variable \$\$ALinkKeys.

As **Mif2Go** processes FrameMaker input for a topic, the \$\$ALinkKeys list gathers keywords from *ALinkTarget* paragraphs until it is time to create the ALink button object for a topic, described in §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315.

See also:

- §9.7.5.3 [Initializing the ALink keyword list counter](#) on page 315
- §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315
- §9.7.5.5 [Positioning the ALink button in each HTML Help topic](#) on page 316

9.7.5.2 Creating a list of ALink keywords from markers

Suppose you use a FrameMaker marker (for example, **Subject**) for ALink keywords. To capture ALink keywords from **Subject** markers, you can remap **Subject** markers to a new marker type **AKey**, and also clone the resulting **AKey** markers; see §29.3 [Remapping marker types and hypertext commands](#) on page 836:

```
[Markers]
Subject=AKey ALink
```

When you remap a **Subject** marker with this assignment, both **AKey** and **ALink** markers get copies of the content of the **Subject** marker.

ALink informs an ALink object

ALink is a predefined custom marker type; see §29.2.1 [Identifying dedicated custom marker types](#) on page 832. **ALink** marker content (inherited from the original **Subject** markers) is used for the ALink Name property of an HTML Help ALink object (*not* the button object, which you will construct with macros); see §7.6.4.1 [Adding related-topic link keywords via markers](#) on page 221.

AKey informs an ALink button

AKey is an *ad hoc* custom marker type; **AKey** markers inherit the content of the original **Subject** markers. Assign **AKey** markers the Code property (see §29.4.1 [Assigning properties to marker types](#) on page 838):

```
[MarkerTypes]
AKey=Code
```

Store marker content in a list variable

Assigning the Code property means that the content of each **AKey** marker (the ALink keyword inherited from a remapped **Subject** marker) can be wrapped in “before” and “after” code:

```
[MarkerTypeCodeBefore]
AKey=<$$Nkeys++><$$ALinkKeys[ $$Nkeys ] = "

[MarkerTypeCodeAfter]
AKey=" >
```

The “before” code does the following:

- Increments a counter, `Nkeys` (which will be initialized to zero before each topic).
- Uses `Nkeys` to index a list variable, `$$ALinkKeys` (see §28.4 [Using multiple-value list variables](#) on page 806).
- Provides an opening double quote for the content of the marker.

The “after” code closes the double quote after the content, and ends the list-variable assignment. The result is code that looks like this:

```
<$$Nkeys++><$$ALinkKeys[ $$Nkeys ] = "keyword">
```

This code stores content taken from the original **Subject** marker in the `Nkeys` slot in list variable `$$ALinkKeys`.

As **Mif2Go** processes FrameMaker input for a topic, the `$$ALinkKeys` list gathers keywords from **Subject** markers until it is time to create the ALink button object for the topic, described in §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315.

See also:

§9.7.5.3 [Initializing the ALink keyword list counter](#) on page 315

§9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315

§9.7.5.5 [Positioning the ALink button in each HTML Help topic](#) on page 316

9.7.5.3 Initializing the ALink keyword list counter

To set the counter for ALink keyword list variable `$$ALinkKeys` to zero at the beginning of each FrameMaker file:

```
[MacroVariables]
Nkeys=0
```

The `[ALinkButton]` macro (see §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315) sets `Nkeys` back to zero again after finishing each button, to re-initialize `Nkeys` for the next topic.

See also:

§9.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 313

§9.7.5.2 [Creating a list of ALink keywords from markers](#) on page 314

§9.7.5.3 [Initializing the ALink keyword list counter](#) on page 315

9.7.5.4 Building an ALink button object from an ALink keyword list

When it comes time to output an ALink button for a topic, the following macro is invoked to process list variable `$$ALinkKeys` (see §9.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 313 or §9.7.5.2 [Creating a list of ALink keywords from markers](#) on page 314) and add each keyword to the button object:

```
[ALinkButton]
<$_if (Nkeys > 0)>
  <$$ALinkButtonStart><$$ALinkParamNum=1>\
  <$_repeat ($$Nkeys)>\
    <$$ALinkParamText=$$ALinkKeys[$$ALinkParamNum]>\
    <$$ALinkParamNum++><$$ALinkButtonParam>\
  <$_endrepeat>\
  <$$ALinkButtonEnd><$$Nkeys=0>\
<$_endif>
```

`[ALinkButton]` uses two additional macro variables:

```
$$ALinkParamNum    Keyword parameter counter (the N in "ItemN")
$$ALinkParamText    Text of a keyword.
```

`$$ALinkParamNum` is initialized to 1, and then incremented *before* each keyword parameter is added to the button object, because the "ItemN" keyword parameters for the button object start with $N=2$, not $N=1$ (see §9.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 312).

`[ALinkButton]` invokes three other macros to build the button object:

[Start of button object:](#) `[ALinkButtonStart]`

[Keyword parameters:](#) `[ALinkButtonParam]`

[End of button object:](#) `[ALinkButtonEnd]`

*Start of button
object*

The first part of the ALink button object is straightforward, and uses the same code described in §9.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 312:

```
[ALinkButtonStart]
<object id="hhctrl" type="application/x-oleobject"
  classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
  width="100" height="100">
  <param name="Command" value="ALink">
  <param name="Button" value="Text:Related topics">
  <param name="Flags" value="1">
  <param name="Item1" value="">
```


Keyword parameters Next come the "Item2" through "ItemN" parameters, which are added to the button object one by one, as list variable \$\$ALinkKeys is processed:

```
[ALinkButtonParam]
<param name="Item<$$ALinkParamNum>" value="<$$ALinkParamText>">
```

End of button object The last piece ends the button object:

```
[ALinkButtonEnd]
</object>
```

This version of the [ALinkButton] macro assumes that each item in the \$\$ALinkKeys list contains a single ALink keyword. To process a list that contains multiple keywords per list item, you would need a slightly more complex version; see §9.7.5.6 [Including multiple ALink keywords in a paragraph or marker](#) on page 316.

See also:

§9.7.5.5 [Positioning the ALink button in each HTML Help topic](#) on page 316

9.7.5.5 Positioning the ALink button in each HTML Help topic

The macro that creates an ALink button object must be invoked after all ALink keywords in a topic have been added to the \$\$ALinkKeys list. The easiest way to ensure that the button follows all sources of keywords is to invoke the [ALinkButton] macro at the very end of each topic. For example:

```
[Inserts]
Bottom=<br><$$ALinkButton>
```

See §18.5 [Inserting HTML code in split and extract files](#) on page 598.

See also:

- §9.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 313
- §9.7.5.2 [Creating a list of ALink keywords from markers](#) on page 314
- §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315

9.7.5.6 Including multiple ALink keywords in a paragraph or marker

An enhanced version of the [ALinkButton] macro (see §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315) parses each \$\$ALinkKeys list item for multiple ALink keywords, allowing you to include several keywords (separated by semicolons) in each *ALinkTarget* paragraph or **Subject** marker.

This version of the [ALinkButton] macro uses two additional macro variables:

\$\$ALinkKeyItem	Counts \$\$ALinkKeys list items (while \$\$ALinkParamNum counts keywords and labels the button-object keyword parameters, as before).
\$\$ItemContent	Holds a copy of each potentially multiple-keyword list item for chopping into individual ALink keywords.

This button macro invokes the same [ALinkButtonStart], [ALinkButtonParam], and [ALinkButtonEnd] macros described in §9.7.5.4 [Building an ALink button object from an ALink keyword list](#) on page 315:

```
[ALinkButton]
<$_if (Nkeys > 0)>
  <$$ALinkButtonStart><$$ALinkKeyItem=1><$$ALinkParamNum=1>\
  <$_repeat ($$Nkeys)>\
    <$$ALinkParamText=$$ALinkKeys[$$ALinkKeyItem]>\
    <$$ALinkKeyItem++><$$ItemContent=$$ALinkParamText>\
  <$_while ($$ItemContent contains ";")>\
```



```

<$$ALinkParamText=($$ItemContent before ";">\
<$$ALinkParamNum++><ALinkButtonParam>\
<$$ItemContent=($$ItemContent after ";">\
<$_endwhile>
<$$ALinkParamText=$$ItemContent><ALinkButtonParam>\
<$_endrepeat><ALinkButtonEnd><$$ALinkParamCount=0>\
<$_endif>

```

See §28.6.4.3 [Using loop structures](#) on page 816 for an explanation of loop controls `$_repeat` and `$_while`.

See §28.6.5 [Specifying substrings in expressions](#) on page 817 for an explanation of string operators `contains`, `before`, and `after`.

9.7.6 Creating buttons for other types of related-topic links

For related-topic links other than ALinks, you would have to add to an `<object>` macro an "ItemN" for each type. The following example includes just one:

```

[ParaStyleCodeReplace]
ALinkUse=<$RelLinkButton>

[RelLinkButton]
<object id="hhctrl1" type="application/x-oleobject"
  classid="clsid:adb880a6-d8ff-11cf-9377-00aa003b7a11"
  width="100" height="100">
  <param name="Command" value="Related Topics">
  <param name="Button" value="Text:Related">
  <param name="Item1" value="testing2;reference.htm">
</object>

```

This is not a good idea, because if the file name changes as a result of a split, you would have to remember to edit the macro by hand; though if you use the macro only to navigate to the start of fixed files, this would not be an issue.

You would have to edit the `[RelLinkButton]` macro to suit your own purposes. See the Microsoft HTML Help on-line documentation, under commands for the ActiveX control, for information about the many parameters you can use. Also see §9.9.8 [Customizing contents and index for HTML Help](#) on page 324 for special **Mif2Go** settings that provide a way to specify contents and index parameters.

9.8 Using secondary windows in HTML Help

When you jump to a secondary window in HTML Help, you get only one instance of that window. Whenever you target the secondary window, the window itself stays in place, and only the content is replaced.

In this section:

§9.8.1 [Defining secondary windows for HTML Help](#) on page 317

§9.8.2 [Jumping from a topic to a secondary window](#) on page 318

§9.8.3 [Jumping from contents or index to a secondary window](#) on page 318

See also:

§7.7 [Jumping to secondary windows in Help systems](#) on page 224

9.8.1 Defining secondary windows for HTML Help

Use HTML Help Workshop to define secondary windows for HTML Help. Be sure to give each window a name that does not exceed eight characters.

If you tell **Mif2Go** to rewrite your .hhp file each time (see §9.3.9 [Regenerating the HTML Help project file](#) on page 301), add definitions of secondary windows to your configuration file. The best way to do this is to define all windows in HTML Help Workshop, save the .hhp file, then re-open it in a text editor such as Notepad, and copy the contents of the [WINDOWS] section here:

```
[HHWindows]
main= ....
SecWin= ....
```

9.8.2 Jumping from a topic to a secondary window

You can use either a character format or a paragraph format to create a hotspot for a jump from a topic to a secondary window. Assign the window name to the hotspot format:

```
[SecWindows]
; doc format = name of secondary window to use for jumps from
; within the span marked by this format (same as WinHelp usage)
HotspotFmt=wndwname
```

If more than one CHM file is involved in a jump, you must also specify how those files are mapped; see §9.15.1 [Interlinking multiple CHM files](#) on page 336.

See also:

§7.7 [Jumping to secondary windows in Help systems](#) on page 224

§9.8.1 [Defining secondary windows for HTML Help](#) on page 317

9.8.3 Jumping from contents or index to a secondary window

To create a jump to a secondary window from contents or index in HTML Help, you can use either a paragraph format or a marker in the target topic. Using a marker allows you to designate only selected topics to be displayed in secondary windows when those topics are accessed from contents or index.

In this section:

§9.8.3.1 [Assigning a secondary window with a paragraph format](#) on page 318

§9.8.3.2 [Assigning a secondary window with a marker](#) on page 319

9.8.3.1 Assigning a secondary window with a paragraph format

To use a paragraph format to force a jump from contents or index to a secondary window, assign property window to the format of the paragraph that is the target of the contents or index entry (*not* to the paragraph format of the entry itself):

```
[HTMLParaStyles]
; Window specifies only that access from the contents or index
; opens the topic in the window named in [StyleWindow]
TopicHeadingFmt=Window
```

Also assign the name of the secondary window to the target paragraph format:

```
[StyleWindow]
; para style = window to use in HH when accessed from contents
; or index
TopicHeadingFmt=wndwname
```

*Jumps from
contents*

A jump from the contents loads a topic in a secondary window only if the *first* paragraph in the topic has a format assigned the window property. However, a **Window** marker in the first paragraph overrides any window property assigned to that paragraph format.

Jumps from index A jump from the index loads the topic in a secondary window if the **Window** paragraph format appears *anywhere* in the topic. For paragraphs after the first in a topic, whichever comes first, **Window** marker or paragraph assigned the **Window** property, determines which secondary window will be used for jumps from the index.

See also:

§9.8.1 [Defining secondary windows for HTML Help](#) on page 317

§9.8.3.2 [Assigning a secondary window with a marker](#) on page 319

9.8.3.2 Assigning a secondary window with a marker

To use a marker to force a jump from contents or index to a secondary window, insert a marker of type **Window** in the target topic. Supply the name of the secondary window as the content of the **Window** marker.

A **Window** marker in the first paragraph of a topic overrides any **Window** property assigned to that paragraph format, for jumps from both contents and index. See §9.8.3.1 [Assigning a secondary window with a paragraph format](#) on page 318.

Jumps from contents

If you place a **Window** marker in the first paragraph of a topic, a jump from the table of contents to that topic will load the topic in the secondary window named in the marker. **Window** markers in subsequent paragraphs are ignored for jumps from contents.

Jumps from index

For paragraphs after the first in a topic, whichever comes first, **Window** marker or paragraph assigned the **Window** property, determines which secondary window will be used for jumps from the index.

See also:

§9.8.1 [Defining secondary windows for HTML Help](#) on page 317

§9.8.3.1 [Assigning a secondary window with a paragraph format](#) on page 318

§29.2 [Adding custom marker types](#) on page 832

9.9 Generating contents and index for HTML Help

Although topics in HTML Help can display special characters, contents and index cannot. Be aware that HTML Help Workshop settings for contents and index are different for stand-alone versus merged CHM files.

In this section:

§9.9.1 [Choosing how to generate HTML Help contents and index](#) on page 319

§9.9.2 [Choosing whether to generate binary contents or index](#) on page 320

§9.9.3 [Generating contents and index with HTML Help Workshop](#) on page 321

§9.9.4 [Generating contents and index with Mif2Go](#) on page 321

§9.9.5 [Configuring contents entries for HTML Help](#) on page 322

§9.9.6 [Providing mid-topic contents links in HTML Help](#) on page 323

§9.9.7 [Making the TOC track index links in HTML Help](#) on page 323

§9.9.8 [Customizing contents and index for HTML Help](#) on page 324

9.9.1 Choosing how to generate HTML Help contents and index

When you first set up an HTML Help project from within FrameMaker, **Mif2Go** assumes you will want **Mif2Go** to generate any contents or index you specified; see §7.3.4.1 [Choosing contents and index methods for HTML-based Help](#) on page 207.

To specify whether contents, index, or both should be generated for HTML Help:

```
[MSHtmlHelpOptions]
; ListType = Both (default), Contents, or Index
```

After you set up your HTML Help project, you can choose whether contents and index will be generated by **Mif2Go** or by HTML Help Workshop, or by neither. This choice is governed by the following setting:

```
[MSHtmlHelpOptions]
; RefFileType = HHW (for HH Workshop), Full (single FrameMaker file),
; Body (FrameMaker book), or None (do not generate).
RefFileType = Full
```

When `RefFileType=Full` or `Body`, **Mif2Go** generates contents and index for HTML Help.

When `RefFileType=HHW`, **Mif2Go** removes attributes (including `class`) from `<Hn>` tags, and leaves the task of generating contents and index to HTML Help Workshop. You can edit the configuration file to change the value of `RefFileType`.

If you remove all settings for `RefFileType` from the HTML Help configuration chain, the default becomes `RefFileType=HHW`; therefore, if you specified that you want contents or index generated, they will be generated by HTML Help Workshop.

The default is `RefFileType=HHW` *only when you use the command-line version of Mif2Go* (see §37 [Converting via DCL](#) on page 995). When you set up an HTML Help project from within FrameMaker, *this default is always overridden*. The **Mif2Go Set Up** dialog always specifies one of the following:

```
RefFileType=Body (for a book, or for a chapter while the book is open)
RefFileType=Full (for a standalone FrameMaker file).
```

See also:

§7.3.4 [Modifying contents or index production for HTML-based Help](#) on page 206
 §9.9.3 [Generating contents and index with HTML Help Workshop](#) on page 321.

9.9.2 Choosing whether to generate binary contents or index

To produce binary TOC and index for HTML Help:

```
[MSHtmlHelpOptions]
; BinaryTOC = No (default) or Yes (required for native browse)
BinaryTOC = Yes
; BinaryIndex = No (default) or Yes (required to merge .chm files)
BinaryIndex = Yes
```

These settings take effect only when `WriteHelpProjectFile=Yes`; see §9.3.9 [Regenerating the HTML Help project file](#) on page 301.

Alternatively, you can specify binary TOC or index generation directly in the `.hbj` file, under `[Options]`; see §9.4.3 [Adding tabs and toolbar buttons to HTML Help](#) on page 303.

There are trade-offs to generating binary navigation features for HTML Help. [Table 9-3](#) lists the pros and cons of specifying a binary TOC or a binary index.

Table 9-3 Binary TOC/Index advantages and disadvantages for HTML Help

Binary feature	Advantages and disadvantages
TOC	<p>Pros: Supports browse via Prev and Next buttons in the HTML Help viewer; see §9.4.3 Adding tabs and toolbar buttons to HTML Help on page 303.</p> <p>Supports no-link entries in the TOC; see §9.9.5 Configuring contents entries for HTML Help on page 322.</p> <p>Allows the TOC to stay synchronized with topics selected via the index; see §9.9.7 Making the TOC track index links in HTML Help on page 323.</p> <p>Cons: Incompatible with merging .chm files at run time; see §9.15.5 Comparing HHW settings for stand-alone vs. merged CHMs on page 339.</p> <p>Can cause problems with mid-topic TOC links; see §9.9.6 Providing mid-topic contents links in HTML Help on page 323.</p>
Index	<p>Pros: Supports merging .chm files at run time; see §9.15.5 Comparing HHW settings for stand-alone vs. merged CHMs on page 339.</p> <p>Cons: Prevents index customization; see §7.5.9 Customizing index sort order on page 216.</p>

9.9.3 Generating contents and index with HTML Help Workshop

Contents If you specify `RefFileType=HHW`, HTML Help Workshop constructs the Contents panel, and uses your `<Hn>` tags as the source of the line items. This has two consequences:

- For HTML Help Workshop to accept them, **Mif2Go** must treat differently the `<Hn>` tags in the body file from which the .hhc file is created, by omitting all attributes, including CSS style; therefore you lose control over much of their appearance in the body file.
- The sequence in which HTML Help Workshop uses the `<Hn>` tags is determined by the sequence of the files listed in the .hhp file; if a file is out of order in the list, its contents appear out of order in the .hhc file. When many of your files are created by splitting, keeping the .hhp list up to date becomes a maintenance nightmare.

Index When HTML Help Workshop creates the index, **Mif2Go** embeds FrameMaker index markers (minus the parts that HTML Help does not understand) as K-type index entries in the .htm files. This has two consequences:

- Selecting an index item does not take you to the place in the file where that item is used; instead, it puts you at the start of the HTML file, even if the item sought is at the end.
- You have no way to control sort order.

It is best *not* to use HTML Help Workshop to modify the contents and index files, but to include these files in the compilation *as is*, for this reason: **Mif2Go** uses the `[sort]` information from FrameMaker to order the entries in the index file. If you regenerate this file using HTML Help Workshop, that work is trashed, and sorting goes by text only. The same is true if you specify a binary index.

Note: A binary index is required if you merge CHM files at run time.

See §7.5.9 [Customizing index sort order](#) on page 216.

9.9.4 Generating contents and index with Mif2Go

If you specify either `RefFileType=Full` or `RefFileType=Body`, **Mif2Go** creates contents or index files or both. See §7.3.4 [Modifying contents or index production for HTML-based Help](#) on page 206.

When you use the .hhp project file to compile a CHM file, **Mif2Go** activates the **Contents** and **Index** tabs in the navigation pane. The sequence of files in the .hhp file does not matter; in fact, you can specify just *.htm, for all topic files.

When **Mif2Go** creates the .hhc file, you can use the align attribute and CSS classes for tags in the body file from which the .hhc is created, and you can include any tags in the contents. You cannot do this if you let HTML Help Workshop create the .hhc file.

9.9.5 Configuring contents entries for HTML Help

Headings that start topics, or to which you assign the Contents property or a contents level, are automatically included in the TOC; see the following:

§7.4.3 [Including contents entries in HTML-based Help](#) on page 209.

§7.4.4 [Setting contents levels for HTML-based Help](#) on page 210.

Even if you assign the Contents property to several heading levels in the same topic, the resulting TOC links always take you to the top of the .htm file that contains the link destinations. You *can* provide true mid-topic links in the TOC, but at a cost; see §9.9.6 [Providing mid-topic contents links in HTML Help](#) on page 323.

*Split at each
heading level*

The best way to ensure that each TOC link goes to an exact destination is to split your FrameMaker document at each heading level. Each heading becomes the start of a topic instead of being in the middle of a topic, and TOC entries synchronize with topic content.

*No-link entries
require a binary
TOC*

To include a paragraph format in the TOC but omit the link:

```
[HTMLParaStyles]
; NoContLink suppresses linkage for its Contents item in MS HTML Help;
; the item remains in the Contents pane, but clicking it does not
; bring up the corresponding topic in the main pane.
ParaFmt=Contents NoContLink
```

You can use this feature to include section headings in the TOC. To make a no-link paragraph appear *only* in the TOC, and not in any topic:

```
[HTMLParaStyles]
ParaFmt=Contents NoContLink Delete
```

Note: If you specify NoContLink but you do not also specify a binary TOC in HTML Help Workshop, the NoContLink entries disappear from the TOC; and so do any parent entries, unless there is at least one split below the parent that does not itself have any NoContLink subentries. This is a limitation of HTML Help.

*Skipped heading
levels*

If your document skips a heading level (for example, a level 3 heading follows a level 1 heading with no level 2 heading in between), HTML Help promotes the level 3 heading to level 2 in the contents. However, HTML Help does not promote additional level 3 headings in the same subgroup: two or more level 3 headings in succession result in the first appearing with a book icon, and the rest with page icons subordinate to the book icon.

To avoid this problem, **Mif2Go** moves the whole hierarchy up one level. In the example in §9.9.4 [Generating contents and index with Mif2Go](#) on page 321, all level 3 headings in the same subgroup that follow a level 1 heading would show the same indent in the contents; however, that indent would not be the same as for level 3 headings that follow a level 2 heading.

See also:

§7.4 [Configuring contents entries for Help systems](#) on page 209

§9.9.6 [Providing mid-topic contents links in HTML Help](#) on page 323

9.9.6 Providing mid-topic contents links in HTML Help

If you provide mid-topic links in the TOC, you lose contents tracking of your current location in the Help system. And if you specify a binary TOC in HTML Help Workshop (which you must do to enable certain HTML Help features), mid-topic entries in the TOC become relatively useless. These are known HTML Help problems; **Mif2Go** cannot fix them.

Why not to include mid-topic links in the TOC

Providing mid-topic links in the TOC is generally not a good idea, for the following reasons:

- You cannot include any HTML Help features (such as built-in browse buttons) that require a binary TOC.
- The HTML Help window is usually small, perhaps six words per line; scrolling around in a multi-topic page can take a long time.
- Loss of synchronization can mystify users.

No binary TOC with mid-topic links

If you specify a binary TOC in HTML Help workshop, and you have mid-topic links in the TOC, the name of the last TOC link to a given topic file becomes the name of *all* links to the file, unless you use the following settings for all but the first heading:

```
[HTMLParaStyles]
Midtopichead=Contents NoContLink
```

However, with this setting the mid-topic entries are no longer active links, which is likely to annoy users.

If you must have mid-topic links in the TOC

If you are willing to give up synchronization to get drill-down, and your project does not require a binary TOC, do the following:

1. In [HTMLParaStyles], assign property `Split` only to H1-level heading formats; assign property `Contents` to other heading formats.
2. Set the following option:


```
[MSHtmlHelpOptions]
; ContentsNamesFileOnly = Yes (default, allows tracking)
; or No (allows direct mid-topic jumps to points within files,
; but disables tracking)
ContentsNamesFileOnly=No
```
3. Avoid HTML Help features that require a binary TOC, and make sure your help project file (.hhp file) does *not* specify `Binary TOC=Yes`.

TOC entries reference points inside .htm files (that is, the links have `#place` suffixes), so you can drill down into the file via the TOC; but TOC entries no longer synchronize with topic content.

9.9.7 Making the TOC track index links in HTML Help

If you specify a binary TOC for a stand-alone HTML Help project, you can make the TOC stay synchronized with topics selected via the index. If you intend to merge CHM files, see §9.15.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 339.

To make the TOC track topics accessed from the index:

1. Open the .hhp file in HTML Help Workshop.
2. On the **Files** tab in the *Options* dialog delete the file name under **Index file**.
3. On the **Files** tab in the *Window Types* dialog, make sure each window you have defined references the .hhk file.
4. Compile the project.

You should notice that you can select any index entry (even entries with mid-topic links) and the TOC will track the topic pane.

9.9.8 Customizing contents and index for HTML Help

Two settings allow you to specify additional properties for contents and index:

```
[MSHtmlHelpOptions]
; Properties for the .hhc and .hhk files; can contain macros.
HHCProperties=<param name="ContentsParamName" value="ParamValue">
HHKProperties=<param name="IndexParamName" value="ParamValue">
```

Each setting assigns a parameter for an HTML Help contents or index properties object.

You can use **Mif2Go** macros to assign multiple parameters. For example:

```
[MSHtmlHelpOptions]
HHCProperties=<param name="ContentsParam" value="ContentsValue">
HHKProperties=<$HHKPropMacro>

[HHKPropMacro]
<param name="IndexParam" value="IndexValue">
<param name="OtherIndexParam" value="OtherIndexValue">
```

Mif2Go supplies the enclosing `<object type="text/site properties">` tag.

*Copy Workshop
parameters*

You can choose contents or index properties in HTML Help Workshop, then specify those same properties in the configuration file, so they will be applied every time you run the conversion. For example, to customize contents:

1. Use **Mif2Go** to convert your document to HTML Help.
2. Open the HTML Help project file (*myproj.hhp*) in HTML Help Workshop.
3. With the **Contents** tab selected, click the **Properties** icon.
4. Set whatever properties you wish in the *Table of Contents* dialog, then click **OK**.
5. Save the project, then exit HTML Help Workshop.
6. Open the HTML Help contents file (*myproj.hhc*) in a text editor such as Notepad, and find the properties object at the start of the body. For example:

```
<!-- Sitemap 1.0 -->
</HEAD><BODY>
<OBJECT type="text/site properties">
    <param name="Window Styles" value="0x800425">
</OBJECT>
```

7. If there is just one `<param ...>` tag, copy the tag and assign it as follows:

```
[MSHtmlHelpOptions]
HHCProperties=<param name="Window Styles" value="0x800425">
```

If there is more than one `<param ...>` tag, use a macro; for example:

```
[MSHtmlHelpOptions]
HHCProperties=<$MyHHCProps>

[MyHHCProps]
<param name="Window Styles" value="0x800425">
<param name="Background" value="0x808040">
```

See HTML Help Workshop Help for information about the properties you can specify.

See also:

§7.4 [Configuring contents entries for Help systems](#) on page 209

§7.5 [Configuring index entries for Help systems](#) on page 211

9.10 Converting generated files for HTML Help

Mif2Go generates HTML Help contents and index entries from your FrameMaker TOC and index markers. However, you might want to include additional generated files in your HTML Help output, such as lists of tables and figures (or other special-purpose paragraph lists), and lists of markers (or other special-purpose indexes).

In this section:

§9.10.1 [Converting lists of paragraph references](#) on page 325

§9.10.2 [Converting lists of marker references](#) on page 325

9.10.1 Converting lists of paragraph references

For lists of references to paragraphs, you must make sure the ObjectIDs of all the referenced paragraphs are preserved in the output (see §19.5.3 [Including ObjectID anchors as link targets](#) on page 620), with this setting:

```
[HTMLOptions]
ObjectIDs=All
```

To make the list entries themselves into links, assign the [HTMLParaStyles]ParaLink property to the list-entry paragraph format; see §5.10 [Creating hotspots for hypertext links](#) on page 138.

To suppress page numbers, on the FrameMaker Reference page for the generated file you can apply a character format to <\$pagenum> and to all its leading tabs, then assign that character format the [HTMLCharStyles]Delete property; see §13.8.2.3 [Eliminating page numbers from generated lists](#) on page 445.

For example:

```
[HTMLParaStyles]
FigLOF=ParaLink

[HTMLCharStyles]
LOFpgnum=Delete
```

In the HTML Help version of the **Mif2Go User's Guide**, these features are used to make the Figures and Tables entries into links.

9.10.2 Converting lists of marker references

A list of references to markers can include links to multiple markers for each entry, so you cannot use the [HTMLParaStyles]ParaLink property (see §5.10 [Creating hotspots for hypertext links](#) on page 138) to make the text of the entry the link. Instead you can either use the original page numbers, or substitute a symbol or graphic.

To substitute something else for page numbers, you must apply a character format to <\$pagenum> on the FrameMaker Reference page for the marker list, and assign to that format the [HTMLCharStyles]CodeReplace and [HTMLCharStyles]KeepLink properties; see §13.8.1.3 [Replacing page numbers with symbols or images](#) on page 442.

For example:

```
[HTMLCharStyles]
IOMpgnum= KeepLink CodeReplace
```

You specify HTML code for whatever you want instead of a page number. For example:

```
[CharStyleCodeReplace]
IOMpgnum=<b></b>
```

In the HTML Help version of the **Mif2Go User's Guide** this setting is used to replace the page numbers in keyword indexes with small book icons.

9.11 Providing full-text search (FTS) for HTML Help

For HTML Help, full-text search is created as part of compiling a .chm with HTML Help Workshop; see §9.14 [Compiling and testing HTML Help](#) on page 333. With default settings, you get FTS automatically when **Mif2Go** generates HTML Help.

Note: The HTML Help FTS is built entirely by the Microsoft compiler, and stored in an undocumented binary format within the .chm file. Omni Systems cannot do anything about problems you encounter with its operation.

Omitting FTS To prevent indexing for full-text search in HTML Help:

```
[MShtmlHelpOptions]
; UseFTS = Yes (default) or No (affects Help Project File rewrite)
UseFTS = No
```

Specifying FTS in the .hhp file

The .hhp file for your project contains the setting for FTS, which **Mif2Go** includes by default while creating the .hhp file for you. If you create the .hhp some other way, you must make sure the .hhp file (*not* the **Mif2Go** configuration file) includes the following setting:

```
[OPTIONS]
Full-text search=Yes
```

Including topic titles in search results

For HTML Help to list the names of topics when a user clicks **Search**, you must specify titles for all topics, normally by assigning the Title property to the formats for headings at which you split FrameMaker files to create HTML Help topics. For example:

```
[HTMLParaStyles]
Heading1 = Split Title Contents
Heading2 = Split Title Contents
```

See §18.4.2 [Specifying page titles for split or extract files](#) on page 594.

Indexing for FTS in another language

If you are preparing HTML Help in another language, you must run the compiler, which builds the FTS index, in the target locale. See §9.14.3 [Compiling in a different language](#) on page 335.

Excluding a topic from FTS

Mif2Go excludes a topic from full-text search by changing the topic file extension to .xhtml, even though the file is not actually XHTML. Only files with names containing the string .htm* get indexed (by HTML Help Workshop) for full-text search in HTML Help.

To exclude content from full-text search in HTML Help, insert a **Search** marker with the value No in FrameMaker content you want excluded from search. The value in effect at the end of each split file determines what happens for that file. The value is reset to Yes at the start of each split file.

9.12 Setting up CSH for HTML Help

Producing CSH for HTML Help requires:

- CSH destination identifiers in your document
- links from an application program
- usually, a map file and an alias file to connect links to their destinations.

In this section:

§9.12.1 [Inserting CSH destinations in your document](#) on page 327

- §9.12.2 [Determining whether you need map and alias files](#) on page 328
- §9.12.3 [Specifying and generating a map file for CSH links](#) on page 329
- §9.12.4 [Creating an alias file for CSH links](#) on page 330
- §9.12.5 [Understanding alias-file entries](#) on page 330
- §9.12.6 [Producing a list of aliases and associated topic titles](#) on page 331

See also:

- §7.10.1 [Understanding how CSH works](#) on page 240
- http://helpware.net/htmlhelp/how_to_context.htm.

9.12.1 Inserting CSH destinations in your document

To insert CSH destinations in your FrameMaker document, use one of the following:

- **markers**: for stand-alone Help systems and mid-topic destinations
- **special paragraphs**: for pop-up help accessed only via the application.

Markers are preferred for stand-alone Help systems, and markers provide the only way to insert mid-topic CSH destinations.

If you are creating pop-up context-sensitive help that will serve no other purpose—that is, content will not be accessed except through links from the application—you can use special paragraph formats.

In this section:

- §9.12.1.1 [Using markers for CSH destinations](#) on page 327
- §9.12.1.2 [Using special paragraphs for CSH destinations](#) on page 328

9.12.1.1 Using markers for CSH destinations

To provide a CSH destination with a marker:

1. Place a hypertext **newlink** marker (command **Specify Named Destination** in the FrameMaker *Hypertext* dialog) in the text of your document where you want to display context-sensitive help; see §34.1.2 [Using markers to add links and instructions](#) on page 935. Markers can be anywhere in the text; a good place is at the start or end of the heading for the topic. Each marker must be within the material you want presented to the user.
2. Make the content of the marker a symbolic ID: a unique name with a prefix you specify in the configuration file; see §9.12.4 [Creating an alias file for CSH links](#) on page 330.

Mid-topic destinations

Even if you insert the marker somewhere in the middle of a topic, clicking the associated button in the application takes the user to the beginning of the topic. However, you can provide mid-topic destinations by setting the following option in the configuration file:

```
[MSHtmlHelpOptions]
; UseAliasAName= No (default),
; or Yes (to allow midtopic jumps for CSH)
UseAliasAName=Yes
```

First CSH link is to start of topic

When UseAliasAName=Yes, every CSH link *except the very first* goes directly to a mid-topic destination. Because of a defect in HTML Help, the CSH link for the first symbolic ID in your document always takes the user to the *beginning* of the topic that contains the relevant marker. If this is not acceptable, you can provide a dummy first entry by inserting a **newlink** marker containing a dummy symbolic ID at the start of your FrameMaker document. Also arrange for a dummy link to this destination; see §9.12.3 [Specifying and generating a map file for CSH links](#) on page 329.

See §9.12.5 [Understanding alias-file entries](#) on page 330.

9.12.1.2 Using special paragraphs for CSH destinations

You can use special paragraphs for CSH destinations if you are creating content that will be used *only* for pop-up context-sensitive help; that is, help that will serve no other purpose: content will not be accessed *except* through links from the application.

To use special paragraphs:

1. Create a special paragraph format (for example, *CSHFile*) to use solely for CSH destinations.
2. Insert a paragraph in format *CSHFile* immediately before each help-content entry in your FrameMaker document.
3. Make the text of each *CSHFile* paragraph the name of an *.htm* file to be generated for the entry. *File names must be unique in your document.*
4. Add the following setting to your project configuration file:

```
[HTMLParaStyles]
CSHFile=Filename Title Split Delete
```

Mif2Go starts a new *.htm* file at each *CSHFile* paragraph in your FrameMaker document. Each new *.htm* file has the name you specified in the *CSHFile* paragraph that starts that file. The title of each *.htm* file is the same as the file name; however, the title will not display, because property *Delete* excludes the content of the *CSHFile* paragraph from the *.htm* file.

9.12.2 Determining whether you need map and alias files

Whether you need map and alias files depends on the following:

- the type of API call the developers use in the application program
- the type of CSH destination you use in your document.

[Table 9-4](#) summarizes the most likely circumstances.

Table 9-4 Map and alias files needed for CSH in HTML Help

CSH destination type	Application call type	Link type	Files needed
Marker	HH_HELP_CONTEXT	Numeric ID	Map and alias
Special paragraph	HH_DISPLAY_TOPIC	<i>.htm</i> file name	Alias

Markers If you use markers for CSH destinations, you need both a map file and an alias file:

- map file* Associates each numeric ID in the application with a symbolic ID in your document, where the symbolic ID is the relevant marker text.
- alias file* Associates each symbolic ID in your document with the *.htm* file where the relevant marker is located.

When you use markers, developers should use the *HH_HELP_CONTEXT* API, and specify topics by numeric ID. You need the map and alias files to associate their numeric IDs with your symbolic IDs. Each time you convert your FrameMaker document to HTML Help, Mif2Go uses the symbolic IDs to generate an alias file; see §9.12.4 [Creating an alias file for CSH links](#) on page 330. Usually the developers provide the map file.

When developers use the *HH_HELP_CONTEXT* API, the application must send a numeric ID, and you must have the map file in your project to interpret the numeric ID. The developers need only the map file. You need only to add to your HTML Help project file

the name of the map file and the name of the alias file, before compiling. You can do this via configuration setting; see §9.12.3 [Specifying and generating a map file for CSH links](#) on page 329.

Special paragraphs

If you use special paragraphs for CSH destinations, developers should use the `HH_DISPLAY_TOPIC` API, and specify topics by file name. You do not need an alias file or a map file. Calls from the application program reference the file names you put in the special paragraphs directly, via the compiled help file; check HTML Help Workshop for the correct syntax.

9.12.3 Specifying and generating a map file for CSH links

If the developers of the application for which you are providing context-sensitive help use `HH_HELP_CONTEXT` and specify topics by ID number, ask them for the file that maps symbolic IDs to numeric IDs. (You cannot go directly from numbers to files; you have to go through the symbolic names used in the map and alias files.) For C or C++, the map file is usually named `resource.h`, and contains entries such as the following:

```
#define IDH_Export 1090
#define IDH_CnvDsgnr 1080
```

The map file must be named in the `[MAP]` section of your `.hhp` file, and must be located in or below the directory that contains your `.hhp` file. For example:

```
[MAP]
#include "resource.h"
```

Quotes are required around each `#included` file name.

Instead of editing the `[MAP]` section of your `.hhp` file, you can specify the file name in a configuration setting; and you can have **Mif2Go** generate an initial map file for you.

Specify a map file

To specify the name of the map file:

```
[MShtmlHelpOptions]
; CshMapFile = name of file to #include in .hhp [MAP] for CSH support
CshMapFile = resource.h
```

This way you will not lose the information if **Mif2Go** rewrites the `.hhp` file. However, if you need to reference more than one map file, you must specify any additional map files in the `.hhp` file, and you must prevent **Mif2Go** from rewriting the `.hhp` file. See §9.3.9 [Regenerating the HTML Help project file](#) on page 301.

Generate a map file

While the map file normally comes from the developers, it might be necessary for the writer to produce the first one, to let the developer know what IDs are available.

To have **Mif2Go** generate a map file for CSH links:

```
[MShtmlHelpOptions]
; MakeCshMapFile = No (default) or Yes (generate a map file)
MakeCshMapFile = Yes
; CshMapFileNumStart = Starting number for numeric IDs, default 10000
CshMapFileNumStart = 10000
; CshMapFileNumIncrement = Increment between values, default 10
CshMapFileNumIncrement = 10
```

Mif2Go creates a map file of the name you assign to `CshMapFile`, overwriting any existing file of the same name, and assigning an incremental numeric ID to each of the symbolic IDs included in your document.

9.12.4 Creating an alias file for CSH links

When you use markers for CSH destinations, by default **Mif2Go** generates an alias file for you, named after your document, with extension `.hha`; for example, `MyDoc.hha`.

Mif2Go also creates an entry for the alias file in the HTML Help project file; for example:

```
[ALIAS]
#include "MyDoc.hha"
```

The alias file must be located in or below the directory that contains your `.hhp` file. The alias file contains an entry for each symbolic ID in your document that:

- occurs in a **newlink** marker, and
- begins with one of the prefixes you specify in the configuration file.

Alias prefixes To specify prefixes for symbolic IDs:

```
[MShtmlHelpOptions]
; AliasPrefix = all prefixes wanted in alias file, comma or space
; delimited; if omitted, all newlinks are included
; NOTE: wildcards do not work in prefixes
AliasPrefix=HIDC_, IDH_
```

With this setting, the alias file would include the content of every **newlink** marker in your document that contains a name prefixed with `HIDC_` or `IDH_`. See §9.12.5 [Understanding alias-file entries](#) on page 330 for examples.

No alias file To prevent **Mif2Go** from creating an alias file:

```
[MShtmlHelpOptions]
; MakeAliasFile = Yes (default, make list of newlinks and files) or No
MakeAliasFile=No
```

When `MakeAliasFile=No`, **Mif2Go** does not generate an alias file. In that case, if you are using markers for CSH destinations, you must create the alias file manually, and manually insert the corresponding entry in the HTML Help project file.

When `MakeAliasFile=Yes`, but your document contains no **newlink** markers that qualify, **Mif2Go** does not generate an alias file.

9.12.5 Understanding alias-file entries

By default **Mif2Go** generates alias-file entries of the following form:

```
symbolic_ID=helptopicfile.htm
```

For example:

```
IDH_CnvDsgnr=02x998989.htm
IDH_Export=02x999005.htm
```

Mid-topic destinations

To make a CSH link take the user directly to a mid-topic destination, the alias-file entry for the symbolic ID must include a hash value after the file name:

```
symbolic_ID=helptopicfile.htm#symbolic_ID
```

For example:

```
IDH_110100=ac960367.htm#IDH_110100
IDH_110200=ac960367.htm#IDH_110200
```

To direct **Mif2Go** to generate alias-file entries of this form, specify the following option:

```
[MShtmlHelpOptions]
UseAliasAName=Yes
```

See §9.12.1 [Inserting CSH destinations in your document](#) on page 327.

*First entry cannot
have a mid-topic
destination*

There is a catch: because of a defect in HTML Help alias-file processing, the very first entry in the alias file must *not* have a hash value. Even when you specify `UseAliasAName=Yes`, **Mif2Go** omits the hash value for the first entry; therefore, the CSH link for the first symbolic ID listed in the alias file always takes you to the beginning of the topic that contains the relevant destination. If this is not acceptable, you can provide a dummy first entry by inserting a **newlink** marker containing a dummy symbolic ID at the start of the first file in the book. This symbolic ID must also appear in a valid entry in the map file, so you might have to get the developers to add a corresponding dummy entry to the map file.

Even with this workaround, HTML Help Workshop will report an error on every alias with a hash value; but the CSH links work anyway.

9.12.6 Producing a list of aliases and associated topic titles

To direct **Mif2Go** to prepare a list of all the CSH IDs used in your document, along with the titles (not the file names) of the topics in which each was found:

```
[MShtmlHelpOptions]
; AliasTitle = No (default) or Yes (generate .hht file with titles
; for all topics containing CSH aliases, like the .hha but with titles
; not filenames)
AliasTitle = Yes
```

When `AliasTitle=Yes`, **Mif2Go** writes to the project directory a file named `MyDoc.hht`, where *MyDoc* is the name of your HTML Help project. Each line in the `.hht` file contains an ID followed by the title of its topic. For example:

```
IDH_ChooseProject "Setting up a Mif2Go project"
IDH_Export "Converting documents"
```

9.13 Generating HTML Help in non-Western languages

HTML Help does not support Unicode well, even in the topic pane where it might appear to do so. Topic content is rendered by the Internet Explorer HTML engine, so the topic pages themselves could use UTF-8. However, the Search function works only on characters that are in the Windows code pages that HTML Help supports. For example, English text in a Japanese file can be found, but Search will not find any Japanese content.

In this section:

- §9.13.1 [Converting from Unicode to Windows code pages](#) on page 331
- §9.13.2 [Specifying locale and language for HTML Help](#) on page 332
- §9.13.3 [Preventing inclusion of Unicode numeric references](#) on page 333
- §9.13.4 [Coping with FrameMaker index-entry conversion defects](#) on page 333

See also:

- §7.5.9.3 [Specifying index sort type and locale](#) on page 218
- §9.3.6 [Deciding whether to compile HTML Help](#) on page 300
- §9.14.3 [Compiling in a different language](#) on page 335

9.13.1 Converting from Unicode to Windows code pages

Mif2Go can convert your document from Unicode to the appropriate Windows code pages for HTML Help, by using the ICU library; see:

<http://site.icu-project.org/>

If you have not already done so, download `icu401.zip` from the Omni Systems Web site. To install the ICU code pages, extract all code-page DLLs from `icu401.zip`, and copy the DLLs to both of the following locations:

- `%OMYSHOME%\common\bin`
- your Windows system directory.

Mif2Go will use these code-page DLLs to prepare your HTML Help output, depending on the locale you specify; see §9.13.2 [Specifying locale and language for HTML Help](#) on page 332.

The complete set of Windows code pages potentially needed for CHMs can be found here: <http://msdn.microsoft.com/en-us/goglobal/bb964654>

For CJK languages, you would need these four:

- 932 (Japanese Shift-JIS)
- 936 (Simplified Chinese GBK)
- 949 (Korean)
- 950 (Traditional Chinese Big5)

9.13.2 Specifying locale and language for HTML Help

To specify locale and language for HTML Help (for example, Japanese):

```
[MShtmlHelpOptions]
; HelpFileLanguage = LCID to put in project file, default is for
; US English.
HelpFileLanguage = 0x411 Japanese
```

This is equivalent to setting the following in your `.hhp` file:

```
[OPTIONS]
Language = 0x411 Japanese
```

Mif2Go supports the following locales:

<u>Decimal</u>	<u>Hex</u>	<u>Language</u>
1033	0x409	English (United States)
1032	0x408	Greek
1049	0x419	Russian
1055	0x41F	Turkish
1029	0x405	Czech (used for Central European)
1041	0x411	Japanese
1028	0x404	Chinese (Traditional)
2052	0x804	Chinese (Simplified)
1042	0x412	Korean

Each of these values sets an associated code page for all output files, and overrides any values specified in the configuration file for the following settings in `[HTMLOptions]`:

```
Encoding      §13.4.3 Specifying character encoding for HTML on page 431
XMLEncoding   §14.3.3 Specifying character encoding for generic XML on
               page 460
```

Getting around a defect in HHW in order to display Help title

When you specify a locale identifier (LCID) other than US English, a defect in HTML Help Workshop prevents your Help-file title from being displayed in the CHM file; instead, the title shows as “HTML Help”. **Mif2Go** provides a default workaround that sets the HTML Help Workshop `Language` option to US English for initial creation of the `.hhp` file. Even if the resulting CHM file will be used in other locales, a setting for

HelpFileLanguage is required to display the value you specified for HelpFileTitle instead of just “HTML Help”.

*Fixed spaces
cannot always be
represented*

Because fixed spaces (such as non-breaking spaces and thin spaces) cannot be represented in some code pages, if you are using (for example) the Japanese locale, **Mif2Go** maps all fixed spaces to the ideographic space, U+3000 (x81 x40), for code page 932. For other characters, see §21.5 [Assigning properties to text formats](#) on page 653.

9.13.3 Preventing inclusion of Unicode numeric references

As a partial workaround for the lack of Unicode support, by default **Mif2Go** includes the original Unicode as numeric character references for characters not in the current code page. Therefore, you will get Unicode for any character that could not be rendered in the code page you specified, unless you set the following option:

```
[HTMLOptions]
NumericCharRefs=No
```

These characters will be viewable, but will not work in Search or in the index. See §13.4.3 [Specifying character encoding for HTML](#) on page 431.

9.13.4 Coping with FrameMaker index-entry conversion defects

If you are working with files converted from an older version of FrameMaker to version 8.0 or later, you might encounter a FrameMaker defect. When FrameMaker converts a pre-8.0 file, it converts the content to Unicode in UTF-8 encoding. However, the index markers are not converted correctly, at least for Japanese and probably for all DBCS encodings (Chinese, Korean). Instead of converting character by character, FrameMaker converts byte by byte, encoding each byte of each double-byte character in UTF-8 individually. This is not valid in any sense, and is not a recoverable error. You will have to correct these problems in FrameMaker.

9.14 Compiling and testing HTML Help

It is best to compile HTML Help in a directory different from your **Mif2Go** HTML Help project directory. You can have **Mif2Go** automatically copy the necessary files to a compilation directory, then run the HTML Help compiler; or, you can include copy commands in the configuration file, and run the HTML Help compiler yourself.

In this section:

§9.14.1 [Directing Mif2Go to run the HTML Help compiler](#) on page 333

§9.14.2 [Copying output files and compiling later](#) on page 334

§9.14.3 [Compiling in a different language](#) on page 335

§9.14.4 [Testing HTML Help generation](#) on page 335

§9.14.5 [Registering your HTML Help system for network use](#) on page 335

9.14.1 Directing Mif2Go to run the HTML Help compiler

When you check **Compile Help** in the **Mif2Go Export** dialog (see §3.6 [Converting documents](#) on page 82), or specify the following options in the configuration file, **Mif2Go** automatically runs the HTML Help compiler after generating output files:

```
[Automation]
WrapAndShip = Yes
CompileHelp = Yes
```

For large projects, you might want to stick with `CompileHelp=No`; then after **Mif2Go** finishes the conversion, compile directly from HTML Help Workshop. Otherwise you might encounter memory limitations when **Mif2Go** tries to run the compiler.

If the HTML Help compiler is not on your system PATH, you must tell **Mif2Go** where to find it. For example:

```
[MSHtmlHelpOptions]
; Compiler = path to hhc.exe, not required if its directory is in the
; system PATH environment variable.
Compiler = D:\hh\hhc
```

You might also want to set the following option, so you can see any error messages that result:

```
[Automation]
; KeepCompileWindow = No (default)
; or Yes (so any error messages can be seen)
KeepCompileWindow = Yes
```

When `KeepCompileWindow=Yes`, a system window opens when the compiler runs. If there are no compilation errors, you will see only a command prompt when compilation finishes. You must dismiss the window before **Mif2Go** can continue processing.

If the compiler does not run when `CompileHelp=Yes`, try copying `hh.exe` and `hhc.exe` to `C:\Windows`, and `hha.dll` to `C:\Windows\System32`. Then restart FrameMaker before you run **Mif2Go** again.

To have **Mif2Go** copy the `.hhp` file to another directory for compiling, specify the following:

```
[Automation]
WrapAndShip=Yes
; WrapPath = path to dir for compiling and distribution,
; default is output dir
WrapPath = .\help
```

See §35.6 [Assembling files for distribution](#) on page 961.

See also:

§7.2.4 [Compiling and distributing Help systems](#) on page 204

§9.3.10 [Locating graphics files for HTML Help](#) on page 302

§9.14.2 [Copying output files and compiling later](#) on page 334

9.14.2 Copying output files and compiling later

If you do not want **Mif2Go** to automatically copy output files and run the HTML Help compiler, you can create a separate directory for compilation, then *copy* (do not *move*) the required files to that directory: `.htm`, `.hhk`, `.hhc`, `.hha`, and `.hhp`. Because the project directory contains a lot of other files that are all part of the conversion machinery, you must keep a set of these files in the project directory. Also, place your CSS file in the compilation directory.

Note: If you manually copy files to another directory for compilation, *do not* check **Compile Help** in the **Mif2Go Export** dialog; that option works only on files in the directory specified by `[Automation]WrapPath` (see §35.3 [Understanding path values for deliverables](#) on page 957). Compile from HTML Help Workshop instead.

You can use macros (see §28 [Working with macros](#) on page 787), and on some systems you can use system commands (see §34.4 [Executing operating-system commands](#) on

page 937), to move just the compilable HTML Help files into place after converting a FrameMaker document.

For example, you could define macro <\$SetUpHHDirs> to create the compilation directory at the start of conversion, and macro <\$CopyHHFiles> to actually copy files to the compilation directory after conversion. You would add the following settings:

```
[Automation]
SystemStartCommand = <$SetUpHHDirs>
SystemEndCommand = <$CopyHHFiles>
```

If you copy graphics files to the compilation directory, set the following option in project configuration file `m2htmlhelp.ini`, to remove path information from references to the graphics files:

```
[Graphics]
StripGraphPath = Yes
```

When **Mif2Go** finishes converting your document, select the `.hhp` file in HTML Help Workshop, and compile the project.

9.14.3 Compiling in a different language

To compile HTML Help in a language for a locale other than your current Windows locale, download free command-line utility `SAppLocale` from SteelBytes:

<http://www.steelbytes.com/?mid=45>

`SAppLocale` allows you to run another executable as if you are using a different Windows locale. For example, to compile Japanese HTML Help, you would specify:

```
SAppLocale 1041 path\to\hhc.exe MyProj.hhp
```

The number 1041 is the decimal code for Japanese. To see a list of all the locales, run `SAppLocale` with no parameters. This utility is a must-have for languages that do not use code page 1252. For Korean and Simplified Chinese output, index entries might be corrupted. In that case you would need to compile on a machine with the correct locale specified.

9.14.4 Testing HTML Help generation

You can open HTML Help Workshop and leave it open while you have FrameMaker open. After you save using **Mif2Go** from FrameMaker, switch to HTML Help Workshop to load the resulting `.hhp` file, and click **Compile** (you do not need to **Save** first). When HTML Help Workshop is finished, click **View** to see what you produced. If you find a problem, go back and fix it in FrameMaker (or in the configuration file), rerun **Mif2Go**, and recompile.

Note: If you change a setting in the `.hhp` file, you must close the `.hhp` file in HTML Help Workshop and reopen it before compiling.

9.14.5 Registering your HTML Help system for network use

To use HTML Help over a network when the CHM file is installed on an individual computer, you must register the CHM file in the Windows Registry. This is because current Microsoft security features block HTML Help files viewed from a network drive.

You can use a free tool, `HHReg` from EC Software, to register each CHM file in the Windows registry:

http://www.ec-software.com/products_hhreg.html

CHM files viewed from local drives are not blocked by these security features.

9.15 Mapping and merging CHM files

You can create an HTML Help system that consists of multiple CHM files with interfile links. If all the files are always present, map them. On the other hand, if you are creating a modular Help system, one or more CHM files can be merged into a main CHM file at run time, based only on whether or not the other files are present.

In this section:

- §9.15.1 [Interlinking multiple CHM files](#) on page 336
- §9.15.2 [Synchronizing TOC references to slave CHM files](#) on page 338
- §9.15.3 [Putting up with a binary index for merged CHM files](#) on page 338
- §9.15.4 [Merging CHM files](#) on page 339
- §9.15.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 339

See also:

- §7.11 [Setting up a dynamic modular Help system](#) on page 241
- §9.6.3 [Linking to external files from compiled HTML Help](#) on page 308
- §9.15.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 339
- §19.6 [Linking to other files and other Mif2Go projects](#) on page 621

9.15.1 Interlinking multiple CHM files

When you create multiple interlinked CHM files, you must specify a mapping for each external file name that is specified in the .hlp file (but not in the current CHM file) to the name of the CHM file that contains the corresponding topic.

In this section:

- §9.15.1.1 [Specifying the default CHM file](#) on page 336
- §9.15.1.2 [Mapping FrameMaker files to CHM files](#) on page 337
- §9.15.1.3 [Requiring Mif2Go to use paths for mapped FrameMaker files](#) on page 337

See also:

- §9.6.3 [Linking to external files from compiled HTML Help](#) on page 308
- §19.6 [Linking to other files and other Mif2Go projects](#) on page 621
- Rob Chandler's Web site: <http://www.helpware.net/htmlhelp/linktochm.htm>

9.15.1.1 Specifying the default CHM file

Tell HTML Help Workshop what CHM file you are using as the default:

```
[MSHtmlHelpOptions]
; DefaultChmFile = name of .chm for project if not in [ChmFiles]
DefaultChmFile = MyProj
```

See §9.3.7 [Naming project and compiled files for HTML Help](#) on page 300.

Mif2Go uses the default CHM file name as the destination for any FrameMaker files that have not been mapped to other CHM file names; see §9.15.1.2 [Mapping FrameMaker files to CHM files](#) on page 337.

To support cross references between the current CHM file and CHM files from projects in other output directories, see §19.6 [Linking to other files and other Mif2Go projects](#) on

page 621. To support interproject hypertext links, see §9.6.3 [Linking to external files from compiled HTML Help](#) on page 308.

9.15.1.2 Mapping FrameMaker files to CHM files

For CHM files other than the default file (see §9.15.1.1 [Specifying the default CHM file](#) on page 336), specify how FrameMaker files should be mapped to the other CHM files:

```
[ChmFiles]
; Original or remapped filename (no ext) = chm filename (no ext)
; overrides default set by [MSHTMLHelpOptions]DefaultChmFile
D:/MyBook/Chapter1 = MyProj
```

These mappings takes precedence over any default mapping of the same FrameMaker files to the default CHM file.

No file extensions

Do not include file extensions in mappings.

Specify paths to FrameMaker files

It is best to include a path to each FrameMaker file, because you could have several files with the same name in different books or projects from which you generate different CHM files. Without file paths, you would have no way to differentiate these files. Although you can use either forward slashes or backslashes in paths to FrameMaker files, forward slashes are preferred. FrameMaker stores cross-reference paths with forward slashes, and **Mif2Go** uses those cross-reference paths to find the referenced files. See §9.15.1.3 [Requiring Mif2Go to use paths for mapped FrameMaker files](#) on page 337.

Multiple paths to a single FrameMaker file

To handle several possible paths to the same FrameMaker file, add a line for each path. For example:

```
[ChmFiles]
MyDoc1 = CHMa
MyDoc2 = CHMb
.../GroupB/MyDoc2 = CHMb
G:/test/GroupB/MyDoc2 = CHMb
```

Avoid paths to CHM files

It is best not to specify paths for CHM files mapped in [ChmFiles], because Microsoft does not allow relative paths to CHM files. Although you can specify an absolute path, absolute paths are not a good idea. You cannot predict where the file will be placed on every system. When no path is specified, HTML Help uses the Windows Registry entry to find the CHM file, provided one of the following is true:

- the CHM file has been used at least once
- the installer created the correct Registry entry for the CHM file.

Multiple FrameMaker books

To link CHM files generated from multiple FrameMaker books in different **Mif2Go** projects, consider using a configuration template; see §4.1 [Working with Mif2Go configuration files](#) on page 91. Specify [ChmFiles] mappings for all books in the template. If a given FrameMaker file appears in more than one book, the project configuration file for each such book must include a [ChmFiles] mapping for that file. The project-specific mapping takes precedence over the mapping in the configuration template.

9.15.1.3 Requiring Mif2Go to use paths for mapped FrameMaker files

When you map FrameMaker files to CHM files (see §9.15.1.2 [Mapping FrameMaker files to CHM files](#) on page 337), by default **Mif2Go** first checks for the presence of a FrameMaker file with the path specified in [ChmFiles]; if the file is not found, **Mif2Go** checks for the file without a path.

To require **Mif2Go** to use the path:


```
[MSHtmlHelpOptions]
; RemoveChmFilePaths = Yes (default) to try to match filenames
;   without their path component in [ChmFiles] after trying with it,
;   or No to require the path (with forward slashes)
;   to be present on the left-side names.
RemoveChmFilePaths = No
```

See also:

§9.15.1.1 [Specifying the default CHM file](#) on page 336

9.15.2 Synchronizing TOC references to slave CHM files

The method described in this section works only for slave CHM files that are never used as stand-alone Help files. A simpler way to ensure TOC synchronization is to set `UseChmInLinks=Yes`, as described in §9.6.2 [Specifying href link syntax for HTML Help](#) on page 308.

When you merge CHM files, to ensure that TOC entries for topics in a slave CHM file are synchronized, when you generate the slave `.hhc` file you can have **Mif2Go** prefix references with master-to-slave text for the value of the `Local` parameter:

```
[MSHtmlHelpOptions]
; ContentsLocalValuePrefix = text to put before file references in
;   .hhc, used in slave .chms that are used only with a master.chm,
;   not alone
ContentsLocalValuePrefix = master.chm:/slave.chm:/
```

For example, if the master is `guide.chm` and the slave is `intro.chm`, you would specify:

```
[MSHtmlHelpOptions]
ContentsLocalValuePrefix = guide.chm:/intro.chm:/
```

A reference in the slave TOC would then look like this:

```
<li> <object type="text/sitemap">
<param name="Name" value="Introduction">
<param name="Local" value="guide.chm:/intro.chm:/Introduction.htm">
</object>
```

9.15.3 Putting up with a binary index for merged CHM files

When you merge CHM files, you are totally dependent on the Help Compiler to sort the index; sort strings do not help, and **Mif2Go** cannot change that. This is true regardless of how you produce the CHM. Creating your own `.hhk` file with the order you want does not work, because the compiler ignores that order when creating the binary sort. The only thing that could affect the binary sort order is the sort code in the Windows OS on the machine where the compiler is run.

For example, to get a binary index sorted for Japanese HTML Help, you would have to compile on a native Japanese system, not just on an English system with a Japanese IME running. In that case, you have handed over control of sort order to Windows; you get what it gives you, and that is that. On the other hand, so does everyone else, so users should be used to it.

The other choice is to build the Help as a single, non-merged project, with variations for each use case if some modules are present and others excluded for specific audiences. That may be the only answer, if you do not like the Windows sort order. With five modules, you would have 120 possible combinations, but perhaps not all of them are really used. And even if they are, disk space is cheap, and you could install just the one needed on a given user's system.

9.15.4 Merging CHM files

To merge CHM files at run time, you must designate one of the projects to be the main project (master); the others are subprojects (slaves). In the configuration file for the main project, in the [HelpMerge] section list the names of all the subproject CHM files to be merged, omitting file extensions. For example:

```
[HelpMerge]
LibRef
AdvModule
HelpOnHelp
```

The merge process includes any subproject's [HelpMerge] data; as a result, any other subprojects specified for merging into a given subproject are also integrated into the main project, allowing any degree of nesting of subprojects.

Place a **HelpMerge** marker in your main-project FrameMaker document for each subproject listed in the [HelpMerge] section, to show where the subproject should be merged into the main project, and to specify a contents level for the top TOC entry for the subproject.

Insert the **HelpMerge** marker between two main-project topics, in either of the following places:

- at the start of the following main-project topic, before any text
- at the end of the preceding main-project topic, after all text, in an otherwise empty paragraph.

The content of the **HelpMerge** marker consists of a single-digit contents level for the top TOC entry, followed by a space, followed by the CHM file name of the subproject, without extension. For example:

```
2 HelpOnHelp
```

For more information about merging multiple CHM files, see **Creating Help > Manage Large Document Sets** in HTML Help Workshop *Help on HTML Help*.

See also:

§7.4.4 [Setting contents levels for HTML-based Help](#) on page 210

§7.5.8 [Specifying index link destinations for HTML-based Help](#) on page 215

§9.15.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 339

§29.2 [Adding custom marker types](#) on page 832

Rob Chandler's Web site: http://helpware.net/htmlhelp/how_to_merge.htm

9.15.5 Comparing HHW settings for stand-alone vs. merged CHMs

You might have to experiment with HTML Help Workshop settings to achieve the best combination of functionality and features for your particular project organization and content. HTML Help appears to be “surprisingly complex and not overly predictable”.

For example, a binary TOC is not usually compatible with merged CHM files. However, see:

[http://msdn.microsoft.com/en-us/library/aa814522\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa814522(VS.85).aspx)

Table 9-5 shows the rationale for certain combinations of settings for three CHM roles.

Table 9-5 Rationale for HHW settings by CHM role

CHM role	HTML Help Workshop settings and their effects
Stand-alone CHM files	<p>[OPTIONS]Binary index=No (for "better index disambiguation")</p> <p>[OPTIONS]Binary TOC=Yes (required for native HTML Help browse navigation; see §9.4.3 Adding tabs and toolbar buttons to HTML Help on page 303)</p> <p><i>Options</i> Files tab: remove .hhk file (to synchronize the TOC with the index when Binary TOC is enabled)</p> <p><i>Window Types</i> Buttons tab: check Prev and Next (for native HTML Help browse navigation)</p>
Slave CHM files in a merged HTML Help project	<p>[OPTIONS]Binary index=Yes (required for merged files)</p> <p>[OPTIONS]Binary TOC=No (not compatible with merged files; however, you might successfully merge slaves that have Binary TOCs, if you sacrifice correct native browse navigation)</p> <p><i>Options</i> Files tab: retain .hhk file (or the indexes will not merge)</p> <p><i>Window Types</i> Buttons tab: clear Prev and Next check boxes (native HTML Help browse navigation does not function correctly in merged files)</p>
Master CHM file in a merged HTML Help project	<p>[OPTIONS]Binary index=Yes (required for merged files)</p> <p>[OPTIONS]Binary TOC=No (or the TOC of the merged project will be a mess)</p> <p><i>Options</i> Files tab: retain .hhk file (or the index will not merge)</p> <p><i>Window Types</i> Buttons tab: clear Prev and Next check boxes (native HTML Help browse navigation does not function correctly in merged files)</p>

[Table 9-6](#) summarizes the settings that should work for stand-alone HTML Help projects and for merged projects.

Table 9-6 HTML Help Workshop settings for stand-alone vs. merged CHMs

HTML Help Workshop setting	Stand-alone CHM	Merged CHMs
[OPTIONS] Binary TOC=	Yes	No
[OPTIONS] Binary Index=	No	Yes
<i>Options</i> : Files tab, Index	Remove .hhk	Retain .hhk
<i>Window Types</i> : Buttons tab	Check Prev and Next	Clear Prev and Next

See also:

§9.15.4 [Merging CHM files](#) on page 339

http://www.helpware.net/htmlhelp/how_to_merge.htm

http://www.helpware.net/FAR/help/dlg_hhpedit_sec.htm

http://www.help-info.de/de/FAR/dlg_hhpedit.htm

10 Generating OmniHelp

Mif2Go generates project-specific data and control files for OmniHelp; basic control files can be downloaded from the Web. This section addresses issues that are specific to generating OmniHelp. HTML settings described in section 13 and sections 18 through 34 apply also. Topics include:

- §10.1 [Understanding how OmniHelp works](#) on page 341
- §10.2 [Setting up OmniHelp viewer control files](#) on page 342
- §10.3 [Setting up an OmniHelp project](#) on page 345
- §10.4 [Using CSS with OmniHelp](#) on page 350
- §10.5 [Customizing OmniHelp display features](#) on page 352
- §10.6 [Choosing navigation features for OmniHelp](#) on page 356
- §10.7 [Configuring contents and index for OmniHelp](#) on page 357
- §10.8 [Providing related-topic links in OmniHelp](#) on page 359
- §10.9 [Jumping to secondary windows in OmniHelp](#) on page 360
- §10.10 [Configuring full-text search for OmniHelp](#) on page 361
- §10.11 [Setting up CSH for OmniHelp](#) on page 364
- §10.12 [Merging OmniHelp projects](#) on page 366
- §10.13 [Assembling OmniHelp files for viewing](#) on page 369
- §10.14 [Deploying OmniHelp](#) on page 370

See also:

- §7 [Producing on-line Help](#) on page 199

To determine which configuration settings will produce the appearance and functionality you want, see also:

- §13 [Converting to HTML/XHTML](#) on page 423
- §18 [Splitting and extracting files](#) on page 585
- §21 [Mapping text formats to HTML/XML](#) on page 645
- §24 [Converting tables to HTML](#) on page 727

10.1 Understanding how OmniHelp works

OmniHelp is an open-source, cross-platform Help system that displays help topics in a way similar to WebHelp or HTML Help. The OmniHelp viewer consists of a set of HTML (or XHTML) and JavaScript files that present the Help content in a tri-pane format, using any browser that meets the following criteria:

- complies with minimum Web standards
- supports framesets
- supports basic CSS1.

OmniHelp output generated from a FrameMaker document consists of the following:

- a set of HTML (or XHTML) topic files
- a set of JavaScript infrastructure files for contents, index, search, related links, and context-sensitive Help.

The display is controlled by a small set (about 40K) of JavaScript files and CSS files.

Help develop OmniHelp Software developers are invited to contribute to the further development of OmniHelp. The OmniHelp project is officially hosted on SourceForge:

<https://sourceforge.net/projects/omnihelp/>

The OmniHelp Design Report describes how OmniHelp was designed and built. You can read it here:

<http://mif2go.com>

Modify OmniHelp Because OmniHelp is not a compiled Help system, you have access to the source code for the viewer, so you can alter its behavior and appearance beyond just the changes you can make by setting **Mif2Go** configuration parameters. ***However, to undertake major modifications, you must be conversant in JavaScript, CSS, and HTML 4.***

Maintain your modifications Modifying OmniHelp JavaScript, CSS, or HTML files in the viewer directory can lead to a maintenance problem: you have to check your modified files against the corresponding files in each new release, and merge the changes, which might not be trivial. If you change any `ohct*.css`, `oh*.js`, or `oh*.htm` files, use a utility such as WinDiff (free from Microsoft) to compare your files to the updated files. Check each release for new variables that you can set in the configuration file, to control features that formerly required edits to the JavaScript files; see § [New information](#) on page 41. Take advantage of any new settings to minimize JavaScript changes.

License OmniHelp OmniHelp is licensed under the LGPL (Library/Lesser General Public License), which permits its use in commercial products (without requiring those products to also be Open Source) as long as OmniHelp source code, including all modifications, is made available to users:

<http://www.gnu.org/copyleft/lesser.html>

10.2 Setting up OmniHelp viewer control files

To view **Mif2Go** OmniHelp output, you will need a set of JavaScript and HTML or XHTML control files. Most of these control files are included in your **Mif2Go** distribution; the rest are generated each time you run **Mif2Go**.

In this section:

§10.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 342

§10.2.2 [Making OmniHelp viewer control files available](#) on page 343

§10.2.3 [Customizing OmniHelp viewer control files](#) on page 343

§10.2.4 [Examining generated control and data files](#) on page 344

10.2.1 Choosing XHTML vs. HTML OmniHelp control files

Your **Mif2Go** distribution includes two sets of control files: one for HTML, one for XHTML. Which one you use depends on the start-up file type you select. The choice between XHTML and HTML for OmniHelp is usually a matter of personal preference or company policy. However, some older browsers might not display XHTML as well as HTML.

To specify XHTML 1.0 instead of HTML 4.01 for the OmniHelp project start-up file:

```
[OmniHelpOptions]
; OHProjFileXhtml = No (default, to make project file HTML 4.01
;   as required by some browsers), or Yes (to make the project file
;   XHTML 1.0)
OHProjFileXhtml=Yes
```

When `OHProjFileXhtml=Yes`, XHTML versions of several OmniHelp viewer control files are needed instead of HTML files. The names of these files begin with `ox` instead of `oh`; see [Table 10-1](#) on page 344.

Note: The value of `OHProjFileXhtml` determines the default value of `OHViewPath`; see §10.13 [Assembling OmniHelp files for viewing](#) on page 369.

10.2.2 Making OmniHelp viewer control files available

Your **Mif2Go** distribution includes the following OmniHelp viewer control-file directories:

`%OMSYSHOME%\common\system\omnihelp\ohvhtm` (for HTML output)

`%OMSYSHOME%\common\system\omnihelp\ohvxml` (for XHTML output)

Choose the control-file directory that matches your choice of start-up project file type (see §10.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 342), and copy all the files to the corresponding local directory, one of:

`%OMSYSHOME%\common\local\omnihelp\ohvhtm` (for HTML output)

`%OMSYSHOME%\common\local\omnihelp\ohvxml` (for XHTML output)

If you modify any OmniHelp viewer files, modify *only* the files in the local directory. Files in the system directory will be overwritten every time you update **Mif2Go**. *Do not rename any of these files.*

Unless you copy viewer files to some other location, you should not need to specify a path to those files. However, if you do put them somewhere other than the local viewer directory, you must specify the path to this other location:

```
[OmniHelpOptions]
; OHViewPath = path to dir containing the OH viewer files
OHViewPath = D:\path\to\ohview\files
```

The default value of `OHViewPath` depends on the value of `OHProjFileXhtml` (see §10.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 342):

<u>OHProjFileXhtml</u>	<u>OHViewPath default value</u>
No	<code>%OMSYSHOME%\common\system\omnihelp\ohvhtm</code>
Yes	<code>%OMSYSHOME%\common\system\omnihelp\ohvxml</code>

When you finish running **Mif2Go**, for viewing your OmniHelp system, copies of the control files must be included in the same final directory as the OmniHelp HTML or XHTML output files; see §10.13 [Assembling OmniHelp files for viewing](#) on page 369.

10.2.3 Customizing OmniHelp viewer control files

[Table 10-1](#) lists the OmniHelp viewer control files. Files that have names that start with `oh` are for HTML output. Files with names that start with `ox` are for XHTML. In [Table 10-1](#), the names of these files are shown as starting with `o?`. All other files listed are included in both archives.

To customize OmniHelp, you can edit control files marked **Yes** under **Edit?** in [Table 10-1](#). If you are a JavaScript expert, you can also edit `.js` files marked **No**. *Edit control files only if necessary.*

If you intend to undertake extensive customization and distribute the results to third parties, you will also need the files in the following directory:

`%OMSYSHOME%\common\system\omnihelp\ohvm2g`

Copy all files from this directory to the following location:

%OMSYSHOME%\common\local\omnihelp\ohvm2g

Modify only the files in the local directory; those in the system directory will be overwritten every time you update **Mif2Go**.

Table 10-1 OmniHelp viewer control files included in the distribution

File type	File name	Content	View?	Edit?	Ref.
CSS	ohctie.css	CSS for IE for navigation panes	Req for IE	Yes	10.4
	ohctn4.css	CSS for NN4 for navigation panes	Req for NN4	Yes	10.4
	ohctn6.css	CSS for Mozilla for nav. panes	Firefox, etc.	Yes	10.4
	ohctrl.css	Generic CSS for navigation panes	Required	Yes	10.4
HTML (?=h) or XHTML (=?x)	o?ctrl.htm	Loader for JavaScript	Required	No	10.3
	o?frame.htm	Frameset	Required	No	10.3
	o?main.htm	Loading... message	Required	No	
	o?merged.htm	Run-time project merging	Optional	No	10.12
	o?nav.htm	Loading... message for IE	Req for IE	No	
	o?navctrl.htm	Another Loading... message for IE	Req for IE	No	
	o?top.htm	Top-navigation-pane loader	Required	No	10.5.1
JavaScript	ohctrl.js	Start-up and interfacing script	Required	No	10.3
	ohframe.js	Frameset script	Required	No	10.3
	ohfts.js	Search presentation script	Optional	No	10.6
	ohidx.js	Index presentation script	Optional	No	10.7
	o?lang.js	Text of error messages	Required	Yes	10.5.5
	ohlangct.js	Text of control labels, etc.	Required	Yes	10.5.5
	ohlangtp.js	Text of button labels	Required	Yes	10.5.5
	ohmain.js	CSS-setting script for topic pane	Required	No	10.11
	ohmerge.js	Script used in ohctrl.htm	Optional	No	10.12
	ohmerged.js	Run-time merging script	Optional	No	10.12
	ohrel.js	Related-topics presentation script	Optional	No	10.8
	ohstart.js	Start-up script for project	Required	No	10.3
	ohloc.js	Contents presentation script	Optional	No	10.7
	ohloc.js	Top-navigation-pane script	Required	No	10.5.1
	ohlogo.jpg	OmniHelp logo	Optional	No	
Image	ohloc*.gif	Icons for expandable TOC view	Optional	No	10.7
	ohvalid?.gif	W3C validation icon	Optional	No	

10.2.4 Examining generated control and data files

When you run **Mif2Go** with OmniHelp as the output type, **Mif2Go** produces additional data and control files, depending on your project settings. These files are listed in

Table 10-2. All are placed in the project directory you specified for your OmniHelp project. *Do not rename or edit any of these files.*

Table 10-2 OmniHelp data and control files generated by Mif2Go

File type	File name	Content	Ref.
Data	<i>myproj_oha.js</i>	Context-sensitive-help entries	10.11
	<i>myproj_ohc.js</i>	Contents entries	10.6
	<i>myproj_ohk.js</i>	Index entries	10.6
	<i>myproj_ohl.js</i>	Related-topics entries	10.6
	<i>myproj_ohs.js</i>	Full-text-search entries	10.10
HTML or XHTML	<i>_myproj.htm</i>	Start-up project file	10.3
JavaScript	<i>myproj_ohx.js</i>	Project settings from <i>m2javahelp.ini</i>	10.3

In addition to the files listed in [Table 10-2](#), when you first set up an OmniHelp project **Mif2Go** optionally generates a CSS file (default name *ohmain.css*) for topic content; see §10.4.1 [Specifying CSS for topics in OmniHelp](#) on page 350.

10.3 Setting up an OmniHelp project

When you set up an OmniHelp project from within FrameMaker, if configuration file *_m2omnihelp.ini* is not already present in the project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

To add or change any of the options described in this section, edit configuration file *_m2omnihelp.ini*, located in the project directory. Edit configuration file *_m2omnihelp.ini* to add or change any of the options described in this section.

In this section:

- §10.3.1 [Creating an OmniHelp project](#) on page 345
- §10.3.2 [Choosing set-up options for an OmniHelp project](#) on page 346
- §10.3.3 [Deciding where to locate configuration settings](#) on page 347
- §10.3.5 [Giving your OmniHelp project a title](#) on page 348
- §10.3.6 [Specifying the starting topic](#) on page 348
- §10.3.7 [Specifying memory requirements](#) on page 348
- §10.3.8 [Removing paths from interfile links for OmniHelp](#) on page 349
- §10.3.9 [Getting OmniHelp supporting files in the right place](#) on page 349

10.3.1 Creating an OmniHelp project

To create an OmniHelp project:

1. Create a project directory for HTML or XHTML files, separate from the directory where your FrameMaker document is located. Optionally, create a subdirectory for graphics files.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
3. Name your OmniHelp project, and browse to the project directory you created in [Step 1](#).
4. Choose output type Cross-Platform OmniHelp and click **OK**.

5. Select options in the *Set Up OmniHelp Project* dialog (see §10.3.2 [Choosing set-up options for an OmniHelp project](#) on page 346).
6. Use a text editor to edit the resulting `_m2omnihelp.ini` configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).
7. **Important:** To make the configuration fit your usage of headings to start topics, pay special attention to sections [HelpContentsLevels] (see §7.2.1 [Checking automatic Help topic assignments](#) on page 203) and [HTMLParaStyles] (see §18.2 [Splitting files](#) on page 586).

10.3.2 Choosing set-up options for an OmniHelp project

When you choose OmniHelp as the output type for a new project, the *Set Up* dialog shown in [Figure 10-1](#) opens. [Table 10-3](#) shows the corresponding settings in the configuration file. *You must edit the configuration file to specify additional options.*

See also:

§3.4 [Choosing project set-up options](#) on page 79

§7 [Producing on-line Help](#) on page 199

Figure 10-1 Set Up OmniHelp Project

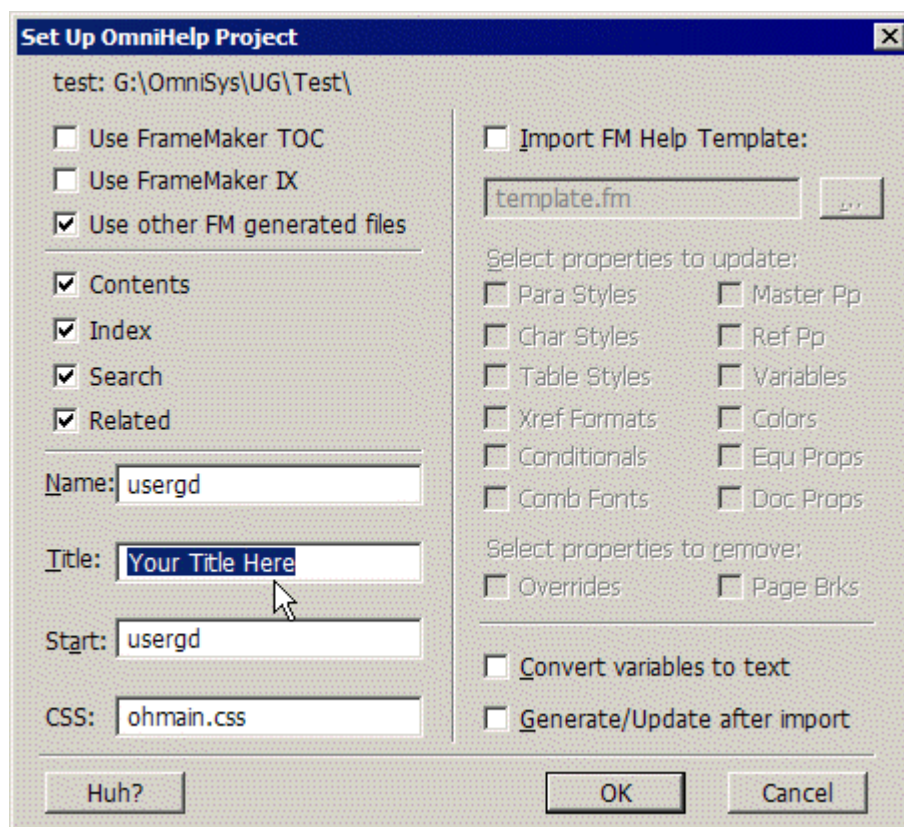


Table 10-3 OmniHelp set-up options and configuration settings

Set-up Option	Configuration file [OmniHelpOptions] section* Setting	Default	Ref.
Contents	NavElems=Toc	Toc Idx Fts Rel	10.6
Index	NavElems=Idx	Toc Idx Fts Rel	10.6

Table 10-3 OmniHelp set-up options and configuration settings (continued)

Set-up Option	Configuration file [OmniHelpOptions] section*		
	Setting	Default	Ref.
Search	NavElems=Fts	Toc Idx Fts Rel	10.6
	UseFts=Yes	Yes	10.10
Related	NavElems=Rel	Toc Idx Fts Rel	10.6
Name	ProjectName=Name	MyDoc	10.3
Title	HelpFileTitle=My Title	Your Title Here	10.3
Start	DefaultTopicFile=FileName	First file in Contents	10.3
CSS	MainCssName=MyStyles.css	ohmain.css	10.4

10.3.3 Deciding where to locate configuration settings

When you set up an HTML Help project from within FrameMaker, if configuration file `_m2omnihelp.ini` is not already present in the project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

Which
configuration file?

To configure HTML Help output, add settings to one of the following files, depending on the desired scope of each setting:

Scope	Configuration file	Location
Current project only	<code>_m2omnihelp.ini</code>	Current project directory
All OmniHelp projects	<code>local_m2omnihelp_config.ini</code>	<code>%omsyshome%\m2g\local\config\</code>

See §30.5 [Deciding which configuration file to edit](#) on page 856.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §13 [Converting to HTML/XHTML](#) on page 423
- §18 [Splitting and extracting files](#) on page 585
- §21 [Mapping text formats to HTML/XML](#) on page 645
- §23 [Including graphics in HTML](#) on page 703
- §24 [Converting tables to HTML](#) on page 727

10.3.4 Naming your OmniHelp project

To specify a name (not a title) for your OmniHelp project:

```
[OmniHelpOptions]
; ProjectName = name for OmniHelp project
ProjectName=myproj
```

The default value is the base name of your FrameMaker book or document. **Mif2Go** uses the value of `ProjectName` for the following purposes:

- generated-data-file base names: `myproj.oh*`
- project identifier, when OmniHelp projects are merged; see §10.12 [Merging OmniHelp projects](#) on page 366.
- project start-up file base name, by default prefixed with an underscore: `_myproj.htm`

To avoid a possible conflict with the name of another file in the same project, you might need to add a prefix, a suffix, or both. You can specify each of the following:

[Project-name prefix](#)

Project-name suffix

*Project-name
prefix*

To specify a prefix for the project name:

```
[OmniHelpOptions]
; OHProjFilePrefix = prefix for project file name so that it does not
; conflict with the name of any file in the project
OHProjFilePrefix=_
```

The default prefix is a single underscore. Although for most purposes you should avoid using any non-alphanumeric characters in file names, just about the only way to make the OmniHelp starting file visible among possibly thousands of HTML files is to force it to sort ahead of all the other files. Prefixing the name with an underscore accomplishes this objective. However, you can specify a different prefix.

*Project-name
suffix*

To specify a suffix for the project name:

```
[OmniHelpOptions]
; OHProjFileSuffix = suffix for project file name so that it does not
; conflict with the name of any other file in the project
OHProjFileSuffix=
```

The default is no suffix at all.

See also:

§10.3.5 [Giving your OmniHelp project a title](#) on page 348

10.3.5 Giving your OmniHelp project a title

To specify a title for your OmniHelp project:

```
[OmniHelpOptions]
; HelpFileTitle = title to put in project-specific frameset file
HelpFileTitle=My Project Title
```

If you do not specify a title, the default title is, literally, “Your Title Here”.

10.3.6 Specifying the starting topic

To specify which topic file to display first, when OmniHelp opens:

```
[OmniHelpOptions]
; DefaultTopicFile = starting topic file name (no extension)
; first file in Contents is used by default
DefaultTopicFile=firstfilename
```

The default starting topic is the first HTML file listed in the generated contents.

10.3.7 Specifying memory requirements

To adjust memory requirements for contents loading:

```
[OmniHelpOptions]
; LowMem = Yes (default, reduce memory requirements or No (faster)
LowMem=Yes
```

When LowMem=Yes (the default) OmniHelp reduces memory requirements while loading the table of contents by writing many short segments instead of one long segment.

For a document that has a long table of contents, you might have to experiment to optimize OmniHelp memory requirements. *Not* reducing memory requirements can speed up contents loading in some browsers, slow it down in others.

10.3.8 Removing paths from interfile links for OmniHelp

Because OmniHelp relies on supporting JavaScript and HTML control files, all HTML output files for an OmniHelp project must reside in the same directory on the target system. Therefore, links between HTML files should not include paths.

Paths are omitted by default from cross-reference and hypertext links:

```
[HTMLOptions]
; RemoveFilePaths = Yes (default, strip hyperlink and xref paths)
; or No
RemoveFilePaths=Yes
```

When RemoveFilePaths=Yes (the default), all HTML output files are assumed to be in the same directory on the target system.

See also:

§19.6.2 [Retaining file paths in interfile links](#) on page 622

10.3.9 Getting OmniHelp supporting files in the right place

Before you can use the OmniHelp viewer, all OmniHelp supporting files must be placed in the same directory structure as the HTML or XHTML output files **Mif2Go** generates from your document. Supporting files include:

[Viewer and control files](#)

[Graphics files](#)

[Optional files.](#)

*Viewer and
control files*

After you run **Mif2Go**, control files and viewer files must be copied from the viewer directory (see §10.2 [Setting up OmniHelp viewer control files](#) on page 342) to the final distribution directory for your project. **Mif2Go** can do this for you; see §10.13 [Assembling OmniHelp files for viewing](#) on page 369.

To view OmniHelp, the view directory must contain the following:

- all files marked **Required** in column **View?** in [Table 10-1](#) on page 344 (including either `oh*.htm` or `ox*.htm`, depending on the start-up file type; see §10.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 342).
- all files listed in [Table 10-2](#) on page 345.

Graphics files

Graphics files must be placed either in the same directory as the generated OmniHelp HTML files, or in a subdirectory. If your graphics files are located elsewhere, they must be copied to the directory with the HTML files, or to a subdirectory.

To tell **Mif2Go** to fetch your referenced graphics:

```
[Automation]
WrapAndShip = Yes
CopyOriginalGraphics = Yes
```

When CopyOriginalGraphics=Yes, **Mif2Go** follows the file paths in your FrameMaker source to find the graphics files to copy.

To tell **Mif2Go** where to put copies of the graphics (for example):

```
[Graphics]
GraphPath = ./graphics
```

The path you specify for GraphPath should be relative to the wrap directory (see §35.3 [Understanding path values for deliverables](#) on page 957). This path will be used in HTML output, as the relative path from the HTML files to their referenced graphics. If you use backslashes in the path, **Mif2Go** converts them to forward slashes before inserting the

references in your HTML output. If you specify `CopyOriginalGraphics=Yes`, **Mif2Go** copies graphics files to the directory specified by `GraphPath`, after generating HTML files.

See also:

§10.13 [Assembling OmniHelp files for viewing](#) on page 369.

§23.3 [Locating graphics files for HTML](#) on page 704

§35.7 [Placing graphics files for distribution](#) on page 965

Optional files A browser loads optional files (marked **Optional** under **View?** in [Table 10-1](#) on page 344), only when you specify the features they support, via configuration settings. Your project might not require all the optional files. For example, if you do not want full-text search, you can omit `ohfts.js` from the OmniHelp view directory; and if you are not merging OmniHelp projects, you do not need `ohmerge*. *` in the view directory.

See §10.2 [Setting up OmniHelp viewer control files](#) on page 342.

10.4 Using CSS with OmniHelp

OmniHelp relies on CSS (cascading style sheets), because **Mif2Go** removes all HTML formatting in the process of generating OmniHelp files. Most likely you will want to provide your own CSS to govern the appearance of text displayed in the topic frame.

In this section:

§10.4.1 [Specifying CSS for topics in OmniHelp](#) on page 350

§10.4.2 [Understanding how CSS works in OmniHelp topics](#) on page 351

§10.4.3 [Specifying CSS for OmniHelp navigation frames](#) on page 352

10.4.1 Specifying CSS for topics in OmniHelp

When you set up a new OmniHelp project (see §10.3.1 [Creating an OmniHelp project](#) on page 345), you can name a default CSS file for the topic frame; the default name of this default file is `ohmain.css`. **Mif2Go** generates `ohmain.css` (or whatever name you specify) and places it in the project directory the first time you convert your document; see §22.3 [Understanding how Mif2Go generates CSS](#) on page 682.

At set-up time **Mif2Go** includes the following CSS-related entries in newly created configuration file `m2omnihelp.ini`:

```
[CSS]
UseCSS=Yes
WriteCssStylesheet=Once
CssFileName=ohmain.css

[OmniHelpOptions]
; CSS default if browser detection fails
MainCssName=ohmain.css
; CSS for main document frame
IECssName=ohmain.css
N6CssName=ohmain.css
N4CssName=ohmain.css
```

That is, all possible OminHelp references to CSS files for the topic frame initially designate the same file. At run time, for text in the topic frame, OmniHelp actually references only the CSS files specified in `[OmniHelpOptions]`, *instead of* the file specified by `[CSS]CssFileName` (if that file has a different name; see §22.4 [Specifying CSS file and link options](#) on page 683).

Different CSS for certain browsers

You can specify (and provide) different CSS files to govern the appearance of text in the topic frame for the following browsers:

- Internet Explorer
- Netscape Navigator 4.x
- Newer versions of Mozilla-based browsers (such as Firefox).

For example:

```
[OmniHelpOptions]
IECssName=ugie.css
N6CssName=ugns6.css
N4CssName=ugns4.css
```

You must also provide a macro to accomplish browser selection; see §22.6 [Linking to alternate CSS files](#) on page 688.

Default CSS for other browsers

If a browser other than those mentioned is being used, OmniHelp looks for a CSS file named `ohmain.css` (or whatever name you specified at set-up), unless you designate a different CSS file for this purpose. For example:

```
[OmniHelpOptions]
MainCssName=general.css
```

The CSS file designated by `MainCssName` is used for topic text when OmniHelp is viewed with browsers other than those for which you specified a different CSS file.

Omit unused CSS

When `MainCssName` designates a file different from the file designated by `[CSS]CssFileName`, the latter file remains in the project directory, and will be copied to the distribution directory (see §35.6 [Assembling files for distribution](#) on page 961), even though it will not be used. And if you remove that file from the project directory, **Mif2Go** will regenerate it the next time you run the project. The only way to permanently eliminate this unused file is to delete it from the project directory and also change the value of the following setting from `Once` to `Never`:

```
[CSS]
WriteCssStylesheet=Never
```

See §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684.

10.4.2 Understanding how CSS works in OmniHelp topics

Each OmniHelp topic file includes in the `<head>` element a `<script>` tag that invokes script file `ohmain.js`. The `ohmain.js` script calls `mainCSS()` in parent-frameset script file `ohframe.js`, which in turn writes a CSS `<link>` into the topic file.

The CSS `<link>` in the topic file specifies the value of `mainCssName`, which is taken from project settings in `myproj_ohx.js` (see [Table 10-2](#) on page 345), which are based on `[OmniHelpOptions]` settings in the configuration file (see §10.4.1 [Specifying CSS for topics in OmniHelp](#) on page 350). Because the `ohframe.js` script detects the browser before writing the `<link>`, the value of `mainCssName` might depend on what you specified in `[OmniHelpOptions]` for `IECssName`, `N6CssName`, or `N4CssName`.

As a result, you can see the effects of CSS in topic text only if both of the following are true:

- the HTML topic file you are viewing was generated by **Mif2Go** for OmniHelp (or you added the proper `<script>` tag to the topic file)
- you are viewing the topic in the OmniHelp frameset, not by itself in a browser.

Otherwise, the CSS `<link>` would not be set.

10.4.3 Specifying CSS for OmniHelp navigation frames

By default, OmniHelp CSS file `ohctrl.css` governs the appearance of text in the top and left navigation frames. You can edit this file to modify class definitions and CSS file names, or you can designate another CSS file for this purpose:

```
[OmniHelpOptions]
; CSS default if browser detection fails
CtrlCssName=ohctrl.css
; CSS for top and left (navigation) frames
IECtrlCssName=ohctrl.css
N6CtrlCssName=ohctrl.css
N4CtrlCssName=ohctn4.css
```

The CSS file designated by `CtrlCssName` is used for top and left navigation frames, but only when the browser with which you view an OmniHelp project is neither a Mozilla-based browser (Netscape, Mozilla, or Firefox) nor Microsoft Internet Explorer.

If you specify a CSS file other than `ohctrl.css` for `CtrlCssName`, be sure the substitute CSS file provides all the default `ohctrl.css` classes.

For an interesting way to accommodate certain CSS differences among browsers, see:

<http://wellstyled.com/css-underscore-hack.html>

10.5 Customizing OmniHelp display features

In this section:

§10.5.1 [Configuring OmniHelp window usage and frameset dimensions](#) on page 352

§10.5.2 [Altering OmniHelp top navigation frame content](#) on page 353

§10.5.3 [Modifying OmniHelp navigation aids](#) on page 353

§10.5.4 [Choosing whether to use cookies for OmniHelp](#) on page 354

§10.5.5 [Localizing the OmniHelp interface](#) on page 354

§10.5.6 [Modifying OmniHelp CSS classes](#) on page 355

§10.5.7 [Modifying the OmniHelp template](#) on page 356

10.5.1 Configuring OmniHelp window usage and frameset dimensions

You can determine whether OmniHelp opens in a new browser window, or in the existing browser window. The default is to open in a new window:

```
[OmniHelpOptions]
; NewWindow = Yes (default, use settings below
;   for FrameHigh, FrameWide, and FrameOptions)
;   or No (use existing browser window)
NewWindow=Yes
```

If you are generating OmniHelp intended for local use, probably you want the OmniHelp frameset to open in a new window, without browser “chrome” (menus, toolbars, icons, and the like). However, see §10.14.3 [Coping with browser quirks](#) on page 371.

*Close empty
window*

By default, the mostly empty browser window that opens initially remains open, behind the OmniHelp window. To close the initial browser window:

```
[OmniHelpOptions]
; CloseOldWindow = No (default)
;   or Yes (if NewWindow, close opening window)
CloseOldWindow=Yes
```

Some browsers ignore the `CloseOldWindow` option (Firefox, Netscape Navigator); others request confirmation before closing the window (Internet Explorer).

*Configure
frameset*

When OmniHelp opens in a new window (the default), you can specify frame dimensions and positioning for the OmniHelp frameset:

```
[OmniHelpOptions]
;Frameset dimensions (in pixels) and properties
;FrameHigh=350
;FrameWide=600
; Frame dimensions, do not reduce any of them at all
;TopHigh=50
;LeftWide=220
;MidHigh=90
; TopFirst = Yes (top frame full width) or No (left frame full height)
TopFirst = Yes
```

Add chrome

You can use JavaScript to add bits of chrome:

```
[OmniHelpOptions]
; FrameOptions = JS window.open() values as in [SecWindows]
```

See §10.9 [Jumping to secondary windows in OmniHelp](#) on page 360. For more information, look up the `window.open()` function in any JavaScript reference.

10.5.2 Altering OmniHelp top navigation frame content

You can use configuration settings to provide HTML or XHTML code for the content of the leftmost and rightmost table cells in the top OmniHelp navigation frame.

The leftmost cell is the same width as the navigation pane below it. Just make sure that whatever you specify for this cell fits in the space above the left navigation pane; bad things happen to the button layout when the buttons do not have enough space.

Note: If you set `TopFirst=No` (see §10.5.1 [Configuring OmniHelp window usage and frameset dimensions](#) on page 352), the leftmost cell is not displayed. The rightmost cell is always displayed.

The (X)HTML code you specify for each of these table cells must be all on one line, and must end with a backslash (`\`). Escape any single quotes in the code by preceding each with a backslash (`\'`). *Do not follow the code line with more than one blank line.*

For example, to substitute your own logo for the Omni Systems logo:

```
[OHTopLeftNav]
; optional (X)HTML content for ohtop nav table left cell
\
```

Or to substitute contact information for the W3C validation button:

```
[OHTopRightNav]
; optional (X)HTML content for ohtop nav table right cell
\
```

To alter other parts of the top navigation frame, you would have to modify JavaScript code in viewer file `ohtop.js`; see §10.1 [Understanding how OmniHelp works](#) on page 341.

10.5.3 Modifying OmniHelp navigation aids

To determine which navigation buttons are displayed in the top navigation pane:

```
[OmniHelpOptions]
; These settings control what buttons are added to the top nav pane
; UseTopButtons = Yes (default, use buttons) or No (use links instead)
UseTopButtons=Yes
```

```

; UseStart = Yes (default, provide Start button) or No
UseStart=Yes
; UsePrevNext = Yes (default, provide Prev and Next buttons) or No
UsePrevNext=Yes
; UseBackForward = Yes (default, provide Back and Fwd buttons) or No
UseBackForward=No
; UseHideShow = Yes (default, provide buttons to hide and show
; the left-side nav pane as in MS HTML Help) or No
UseHideShow=No

```

To hide the left-hand navigation pane when OmniHelp starts:

```

[OmniHelpOptions]
; ShowNavLeft = Yes (default, open with nav pane visible on left)
; or No
ShowNavLeft=No

```

To remove the **List** button from the left-hand navigation pane:

```

[OmniHelpOptions]
; UseListButton = Yes (default) or No (remove from Search panel)
UseListButton=No

```

If you include **Prev/Next** buttons, make sure your TOC does not use mid-topic links, or the **Prev** and **Next** buttons will not work correctly. TOC-level topics should be in their own files. Clicking a mid-topic link in the TOC works as expected, but the **Prev** and **Next** buttons do not; for example, **Prev** takes you back to the previous actual file rather than to the previous item listed in the TOC. This can confuse your users.

10.5.4 Choosing whether to use cookies for OmniHelp

To determine whether users can pick up where they left off in a previous session:

```

[OmniHelpOptions]
; PersistSettings = Yes (default, OH settings persist after closing,
; for the next time the project is re-opened),
; or No (keep during session only)
PersistSettings=Yes

```

*Cookies persist
for a year*

When `PersistSettings=Yes`, the last OmniHelp settings in effect when you exited OmniHelp are stored by the browser as cookies that persist for one year. The next time you open OmniHelp, the same page appears in the same position.

*Delete cookies to
reset this option*

When `PersistSettings=No`, the cookies have no expiration date, which the browser takes to mean “expire at end of session”. However, in the presence of cookies with *later* expiration dates, older cookies do not get replaced by the newer, but stay in effect. This means that once you have opened OmniHelp with `PersistSettings=Yes`, you cannot make the settings desist for a period of one year, except by deleting the cookies.

10.5.5 Localizing the OmniHelp interface

To provide translated equivalents of all OmniHelp button labels and messages, edit the following small JavaScript files:

<code>ohlang.js</code>	Error messages
<code>ohlangct.js</code>	Progress messages, navigation-control labels, default results
<code>ohlangtp.js</code>	Button labels

10.5.6 Modifying OmniHelp CSS classes

Suppose you want a background image behind the entire top OmniHelp panel, buttons and all. To accomplish this you would modify the CSS class **Mif2Go** applies to the top navigation table. JavaScript in `oh_top.js` creates the top navigation pane, and produces (by default) the following HTML for the navigation table:

```
<table class="topnav" border="0" height="50" width="100%">
<tr><td class="topnav" width="230">
&nbsp;OmniHelp &nbsp;</td>
<td class="topnav"><button type="button" id="topStart"
  onclick="parent.ctrl.getStart()"
  title="Go to starting topic">Start</button></td>
... more button cells written here ...
<td class="topnav"></td>
</tr></table>
```

The CSS rules to modify are those for selector `table.topnav`. For example:

```
table.topnav { background-image: url(my_favorite_pic.jpg); }
```

This rule would tile the image to fill the entire top panel.

You would most likely want to add the new rule to all four of the browser-specific CSS files included with OmniHelp:

<code>ohctie.css</code>	Internet Explorer
<code>ohctn4.css</code>	Netscape 4.x
<code>ohctn6.css</code>	Mozilla-derived browsers, such as Firefox
<code>ohctrl.css</code>	Other browsers, such as Opera

If you put the rule in just one of these CSS files, it would be effective only when someone uses that particular type of browser to view your OmniHelp system.

The CSS file for Internet Explorer, `ohctie.css`, has the following top-panel rules; these rules are similar in the other CSS files:

```
/* top panel only */
body.topnav { background: #999 ; margin: 0 }
p.topbody { font: bold 12pt/12pt sans-serif }
table.topnav { vertical-align: middle; border-style: none }
td.topnav { font: bold 10pt/10pt sans-serif;
  margin: 0; text-align: center; vertical-align: top }
```

A body background color is already set, #999999 (gray). You still want a background color; however, you can change its value.

A couple of rules are already present for `table.topnav`, so just add yours:

```
table.topnav { background-image: url(my_favorite_pic.jpg);
  vertical-align: middle; border-style: none }
```

Or, you could add the image to the body rules instead:

```
body.topnav { background-image: url(my_favorite_pic.jpg);
  background: #999 ; margin: 0 }
```

However, do not add the image to both `table.topnav` and `body.topnav`.

Repeat for the other three `ohct*.css` files, and see how the new background looks in different browsers.

10.5.7 Modifying the OmniHelp template

Your **Mif2Go** distribution directory contains a copy of file `ohtpl.ini`, which provides default text values and macros for variable presentation features. *You do not need this file* unless you plan to alter features for which no configuration settings are provided; see §10.1 [Understanding how OmniHelp works](#) on page 341.

You can copy `ohtpl.ini` to your project directory. If you are brave, you can specify a path to `ohtpl.ini` instead of placing it in the project directory; you can even give this template file a different name:

```
[OmniHelpOptions]
; ProjectTemplate = path to template for generating
; OHProj and myproj_ohx.js files, with sections containing text
; and macro references for variable items
ProjectTemplate=ohtpl.ini
```

You need `ProjectTemplate` only when settings are not sufficient; for example, if you undertake a drastic customization of OmniHelp, and add new variables. If you use the same template for all projects, it would be best to keep the template in:

```
%OMSYSHOME%\common\local\omnihelp\ohvm2g
```

Otherwise, keep the template in the project directory.

You can edit a copy of `ohtpl.ini` to experiment with various versions of this template; however, in general you should rarely need to use or modify `ohtpl.ini`.

10.6 Choosing navigation features for OmniHelp

You can choose which navigation features to provide in OmniHelp; the default is to include them all:

```
[OmniHelpOptions]
; NavElems = navigation elements to display in left pane:
; Toc, Idx, Fts, Rel
NavElems=Toc Idx Fts Rel
```

[Table 10-4](#) lists the navigation features; all are displayed in the left-hand frame:

Table 10-4 OmniHelp navigation features

Feature	NavElems value	Reference
Contents	Toc	§7.4 Configuring contents entries for Help systems on page 209 §10.7 Configuring contents and index for OmniHelp on page 357
Index	Idx	§7.5 Configuring index entries for Help systems on page 211 §10.7 Configuring contents and index for OmniHelp on page 357
Full-text search	Fts	§10.10 Configuring full-text search for OmniHelp on page 361
Related topics	Rel	§10.8 Providing related-topic links in OmniHelp on page 359

If you do not intend to include an index, omit the `Idx` item:

```
[OmniHelpOptions]
NavElems= Toc Fts Rel
```

If you do not have `ALinks`, omit the `Rel` item also.

You can choose whether **Mif2Go** generates the data files needed for contents, index, search, and related topics; however, most likely you will never have a reason to change the default settings:

```
[OmniHelpOptions]
; ListType = Both (default), Contents, or Index
ListType = Both
; RefFileType = Full (default for single files),
;   Body (default for books), or None.
RefFileType=Full
```

RefFileType values have the following effects:

- Full *Default for single files.* **Mif2Go** creates a set of *myproj.oh** files for the original FrameMaker file.
- Body *Default for books.* **Mif2Go** creates a set of DCL *.bh** files that are merged with those from other chapter files to produce a combined set of *myproj.oh** files for the book.
- None No *myproj.oh** files nor DCL *.bh** files are produced.

See also:

- §7.3.4 [Modifying contents or index production for HTML-based Help](#) on page 206.
- §10.7 [Configuring contents and index for OmniHelp](#) on page 357

10.7 Configuring contents and index for OmniHelp

In this section:

- §10.7.1 [Understanding OmniHelp contents and index creation](#) on page 357
- §10.7.2 [Choosing whether to use expanding contents or index](#) on page 357
- §10.7.3 [Choosing how far to expand contents and index subentries](#) on page 358
- §10.7.4 [Providing alternate expansion icons for contents or index](#) on page 358
- §10.7.5 [Excluding Open All and Close All buttons](#) on page 359
- §10.7.6 [Redirecting See and See also index entries](#) on page 359

10.7.1 Understanding OmniHelp contents and index creation

Headings that start topics, or to which you assign the `Contents` property or a contents level, are automatically included in the contents for OmniHelp; for details, see:

- §7.4.3 [Including contents entries in HTML-based Help](#) on page 209.
- §7.4.4 [Setting contents levels for HTML-based Help](#) on page 210.

When you click links in the topic pane while the contents pane is displayed, the contents pane stays synchronized with whatever topic you visit.

Mif2Go creates an OmniHelp index from the index markers in your FrameMaker document. As with other HTML-based Help systems, you can specify the granularity of index-link destinations, and customize the sort order of index entries; see §7.5 [Configuring index entries for Help systems](#) on page 211.

10.7.2 Choosing whether to use expanding contents or index

By default, OmniHelp includes an expanding table of contents and an expanding index:

- Click the “+” icon in front of an entry to display subentries; the icon changes to “-”.
- Click the “-” icon to make the subentries disappear; the icon changes back to a “+”.

To collapse lower-level entries so you can browse to other topics via the contents, click the current top topic entry first, then click the “-” icon.

The table of contents always shows you where you are in the topics. While subentries are displayed, you cannot collapse the parent entry unless you first select the parent (or a another entry that is not connected to the subentries); in other words, you cannot close the door while your foot is in it. Click the parent entry first, then collapse the subentries. Otherwise, you would risk losing your place, which is what happens in HTML Help.

You can turn off the expansion feature, so that contents or index entries display fully expanded at all times. Also, some older browsers (for example, Netscape Navigator 4.x) cannot display expanding contents or index, so you would not see the expansion feature even with the default settings.

Omit contents expansion

To omit expanding display of contents subentries, and always display all levels:

```
[OmniHelpOptions]
; TocExpand = Yes (default) or No (do not use expanding TOC)
TocExpand=No
```

Omit index expansion

To omit expanding display of index subentries, and always display all levels:

```
[OmniHelpOptions]
; IdxExpand = Yes (default) or No (do not use expanding Index)
IdxExpand=No
```

10.7.3 Choosing how far to expand contents and index subentries

By default, clicking a “+” icon expands only the subentries at the next level down in the contents or index; any subentries at that level that have subentries of their own remain unexpanded. You can choose how many levels to expand.

Contents expansion levels

To specify how many subentry levels to expand in the table of contents:

```
[OmniHelpOptions]
; TocGroupsOpen = No (default, open TOC with groups closed) or Yes
TocGroupsOpen=No
; TocOpenLevel = level to open to, default 0 for top level only.
TocOpenLevel=0
```

If you set `TocOpenLevel` to a number greater than zero, also make sure that `TocGroupsOpen=No`; otherwise, *all* levels are expanded when you click a “+” icon.

Index expansion levels

To specify how many subentry levels to expand in the index:

```
[OmniHelpOptions]
; IdxGroupsOpen = No (default, open Index with groups closed) or Yes
IdxGroupsOpen=No
; IdxOpenLevel = level to open to, default 0 for top level only.
IdxOpenLevel=0
```

If you set `IdxOpenLevel` to a number greater than zero, also make sure that `IdxGroupsOpen=No`; otherwise, *all* levels are expanded when you click a “+” icon.

10.7.4 Providing alternate expansion icons for contents or index

The contents and index expansion views are displayed with a set of icons (supplied in `ohvNNN.zip`), with names of the form `basenameNN.gif`. The base name for the supplied icons, for both contents and index, is `ohct`. You can replace these icons with a set of your own (or a different set for contents and for index), for example to use with different CSS color schemes.

Contents icons

To specify a base name for an alternate set of TOC expansion icons:

```
[OmniHelpOptions]
; TocIcoBase = ohct (default, basename for set of .gifs used for
```



```
; expanding Toc)
TocIcoBase=myTOC
```

Index icons To specify a base name for an alternate set of index expansion icons:

```
[OmniHelpOptions]
; IdxIcoBase = ohct (default, basename for set of .gifs used for
; expanding Idx)
IdxIcoBase=myIX
```

10.7.5 Excluding *Open All* and *Close All* buttons

By default, OmniHelp provides **Open All** and **Close All** buttons above contents and index, to allow expanding or collapsing all entries with a single click. You can omit these buttons from contents, from index, or from both.

To omit **Open All** and **Close All** buttons from the table of contents:

```
[OmniHelpOptions]
; TocButtons = Yes (default, provide Open All and Close All) or No
TocButtons=No
```

To omit **Open All** and **Close All** buttons from the index:

```
[OmniHelpOptions]
; IdxButtons = Yes (default, provide Open All and Close All) or No
IdxButtons=No
```

10.7.6 Redirecting *See* and *See also* index entries

Mif2Go redirects <\$nopage> *See* and *See also* references for the OmniHelp index. On generating OmniHelp output, **Mif2Go** points each such link to the referenced entry *in the OmniHelp index* (rather than in the topic where the index marker appeared in FrameMaker). The effect is similar to what you can do with IndexRef for other (non-Help) **Mif2Go** output types; see §5.5.4 [Making See and See also index entries into useful links](#) on page 125.

See also:

§7.5.7.1 [Identifying See and See also index references](#) on page 214

§7.5.7.3 [Choosing where to sort See also index references](#) on page 215

10.8 Providing related-topic links in OmniHelp

OmniHelp supports ALink keyword targets and jumps, ALink keyword pools, and KLink jumps to index-link lists. An ALink keyword target or jump can specify multiple ALink keywords, separated by semicolons. An ALink keyword can consist of more than one word; spaces are allowed, but no other punctuation. ALink list links always go to the beginning of a topic; KLink list links should go to the paragraph with the corresponding index marker, just like index entries.

See §7.6 [Providing related-topic links for Help systems](#) on page 219 for ways to include ALink targets and jumps and KLink jumps in your OmniHelp project.

If you include the related-topics feature when you generate OmniHelp (see §10.6 [Choosing navigation features for OmniHelp](#) on page 356), the left navigation pane shows a **Related** tab. When you click an ALink jump hotspot (or click the **Related** tab), OmniHelp automatically switches the left navigation pane to the **Related** tab, and displays a list of links to all other topics to which any of the same ALink keywords are assigned.

You can set up your OmniHelp project to also display a list of the ALink keywords assigned to the topic (via marker or paragraph format) or specified in an ALink jump within the topic:

```
[OmniHelpOptions]
; ShowSubjects = No (default, do not show subjects for ALinks) or Yes
ShowSubjects=Yes
```

The keywords are listed under **Subjects** in the space above the related-topic links.

You can determine whether ALinks go to the beginning of the referenced topic file, or to the beginning of the paragraph that contains the ALink keyword. The default is the beginning of the topic file:

```
[OmniHelpOptions]
; ALinkRefs = File (default) or Para (start of containing para)
ALinkRefs = File
```

See also:

§7.6.2 [Understanding how ALinks work](#) on page 220

§7.6.4 [Adding related-topic link keywords in FrameMaker](#) on page 221

§7.6.5 [Adding ALink and KLink jumps in FrameMaker](#) on page 222

§7.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 224

10.9 Jumping to secondary windows in OmniHelp

To create a jump to a secondary window in OmniHelp, assign the window name to a character or paragraph format. For example:

```
[SecWindows]
; doc format = name of secondary window to use for jumps from
; within the span marked by this style (same as WinHelp usage).
PopWindow=popup, 400, 200
ProcWindow=proc
ProcWin2=proc, 400, 600, menubar=1,titlebar=1,scrollbars=1
```

Window parameters

After the window name you can specify optional comma-separated parameters. The first is width in pixels, the second height in pixels, and the third a list of properties to pass to the JavaScript `window.open()` function. The JavaScript properties are also comma-separated, but unlike the size parameters, JavaScript parameters cannot have spaces between them; see a JavaScript reference for acceptable values.

Pop-up windows

The window name `popup` is reserved for specifying pop-ups, and results in a fresh pop-up window every time. In OmniHelp, a pop-up window persists until you close it; the window does not close when you click inside the pop-up (or click elsewhere), as is the case for pop-ups in other Help systems.

Links from secondary windows

To cause a link *from* a secondary window to bring up a new topic in the *original* topic window (rather than in the secondary window itself), assign reserved window name `main` to the hotspot format. For example:

```
[SecWindows]
Popup=popup, 300, 100
Link2FigWin=figure, 400, 200
Link2Main=main
```

In this example, a regular topic has cross-reference links to a pop-up window and to a secondary window:

- To link to the pop-up topic, character format *Popup* is applied to a hotspot.
- To link to the figure, character format *Link2FigWin* is applied to a hotspot.

In the pop-up topic, character format *Link2Main* is applied to a hotspot for a cross-reference link to a regular topic.

Note: Not all browsers honor the parameters you specify for a pop-up window.

See also:

§7.7 [Jumping to secondary windows in Help systems](#) on page 224

§7.8 [Creating pop-up topics for Help systems](#) on page 225

10.10 Configuring full-text search for OmniHelp

After generating OmniHelp output, by default **Mif2Go** builds a search index: a JavaScript array that lists term and topic number for each non-excluded term that occurs in the content.

In this section:

§10.10.1 [Understanding how OmniHelp FTS works](#) on page 361

§10.10.2 [Generating search data](#) on page 361

§10.10.3 [Making compound terms searchable](#) on page 362

§10.10.4 [Supporting search for non-ANSI text](#) on page 362

§10.10.5 [Specifying length of search terms](#) on page 363

§10.10.6 [Excluding search terms](#) on page 363

§10.10.7 [Excluding content from being searched](#) on page 363

§10.10.8 [Using regular expressions in search](#) on page 363

§10.10.9 [Highlighting search terms found in topics](#) on page 364

10.10.1 Understanding how OmniHelp FTS works

OmniHelp supports single-term and Boolean (AND, OR, NOT) full-text search. A search on a phrase is implemented by successively ANDing the search terms: topics found include all terms in the phrase, except for stop words (see §10.10.6 [Excluding search terms](#) on page 363), whether or not those terms occur together.

There are some limitations:

- Search does not find terms that start with non-alphanumeric characters. For example, to find `$$_currbase`, you would have to search for `currbase`; and to find `-progid`, you would have to search for `progid`.
- Search does not find partial terms; for example, a search for `curr` finds `<$Curr>`, but not `$$_currbase`.
- Search reports every instance of a hit, even if several instances are in the same topic. To remove extra instances of a term from the search index, you can delete duplicate entries from the JavaScript array in `myproj_ohs.js`, either by hand or with a UNIX-style utility such as `uniq`, from Cygwin.

Because OmniHelp is Open Source, anyone can modify or replace the search function to overcome these limitations. You can contribute to the OmniHelp project any tool you make for this purpose, at Sourceforge:

<https://sourceforge.net/projects/omnihelp/>

See §10.1 [Understanding how OmniHelp works](#) on page 341.

10.10.2 Generating search data

Mif2Go generates search data files for OmniHelp by default:

```
[OmniHelpOptions]
; UseFts = Yes (default, write .bhs and myproj_ohs.js files)
; or No (faster)
UseFts = Yes
```

This setting interacts with [OmniHelpOptions]NavElems (see §10.6 [Choosing navigation features for OmniHelp](#) on page 356) as follows:

- If you set NavElems=Fts and UseFts=No, you get a **Search** tab, but it does not work.
- If you do not set NavElems=Fts and you do set UseFts=Yes, you get a **Search** tab, and it works.

If you are not providing full-text search, you can avoid generating search data files by setting UseFts=No, which allows OmniHelp to load faster.

10.10.3 Making compound terms searchable

For compound terms that consist of two words separated by a single punctuation character, you can have the search index include each of the individual terms and also the compound term.

To specify which punctuation characters should be considered in identifying compound terms:

```
[OmniHelpOptions]
; CompoundWordChars = Punctuation marks recognized as connectors of
; compound terms when they separate adjacent words.
CompoundWordChars = :-. _+*
```

The default punctuation characters for compound terms are colon, dash, period, underscore, plus sign, and asterisk.

10.10.4 Supporting search for non-ANSI text

When you type in a search string that contains non-ANSI characters, Windows does not give you UTF-8; it gives you the character in the current code page for the system locale. That is not much of a problem for western European languages, but it does mean that you would need a different search file for each non-Western locale you want to support. Otherwise, the OmniHelp viewer would have to include code-page conversion, which would require a huge library on each Help user's system.

To make sure OmniHelp full-text search finds terms that include non-ANSI characters:

```
[OmniHelpOptions]
; UnicodeFts = No (default, use normal word-break rules for ANSI text,
; or Yes (use the ICU rules for any language including CJK)
UnicodeFts = Yes
; UnicodeLocale = formal identifier of language, default en-US
UnicodeLocale = en-US
```

You will also need two ICU DLL files: icudt40.dll (13 MB) and icuuc40.dll (1 MB). These DLLs are available in archive icu401.zip (6 MB), which you can download from the Omni Systems Web site.

To install the ICU code pages, extract the DLLs from icu401.zip, and copy them to the following locations:

- %OMYSHOME%\common\bin
- your Windows system directory.

When `UnicodeFTS=Yes`, **Mif2Go** will use these DLLs to prepare your OmniHelp output, depending on the value you specify for `UnicodeLocale`.

10.10.5 Specifying length of search terms

You can specify the minimum length of terms to include in the search index; the default is three characters:

```
[OmniHelpOptions]
; SearchWordMin = minimum length of word to index for search,
; default 3
SearchWordMin=3
```

10.10.6 Excluding search terms

When you generate OmniHelp, **Mif2Go** builds a search index that includes all the words in the converted files, except for a list of words to be excluded. **Mif2Go** applies an internal list of words to exclude:

```
[StopWords]
;about after again all already also always and any are been but can
;did does doing each for from has have having its may maybe might not
;see than that the their them then these they this those too use used
;uses using very want was when where which will with would you your
```

You can add your own list of words to exclude, or provide an alternate list, in the `[StopWords]` section of your OmniHelp configuration file.

To specify which list(s) of words to exclude from the search index, set the following option:

```
[OmniHelpOptions]
; UseDefaultStopWords = Yes (use default set, plus any added in your
; own [StopWords]) or No (use your own words only)
UseDefaultStopWords=No
```

When `UseDefaultStopWords=Yes`, **Mif2Go** excludes from the search index the default words and any words listed in the `[StopWords]` section of your configuration file.

When `UseDefaultStopWords=No`, **Mif2Go** excludes *only* words listed under `[StopWords]` in your configuration file.

To augment the `[StopWords]` list after you generate OmniHelp, open `myproj_ohs.js` in a text editor, and copy unwanted words to the `[StopWords]` section of your project configuration file.

10.10.7 Excluding content from being searched

To exclude content from full-text search in OmniHelp, insert a **Search** marker with the value `No` at the beginning of FrameMaker content you want excluded from search, and another with the value `Yes` at the end of content to be excluded. The value in effect at the end of each paragraph determines what happens for that paragraph. The value is reset to `Yes` at the start of each split file.

10.10.8 Using regular expressions in search

You can use JavaScript regular expressions in OmniHelp search, by prefixing the search text with a forward slash (/). To learn the arcane syntax required, consult a JavaScript reference.

Note: Omni Systems provides *no technical support* for this feature.

10.10.9 Highlighting search terms found in topics

When you click a link in the OmniHelp search results list, by default each term found in the target topic is highlighted in yellow.

To turn off search-term highlighting, or change the style:

```
[OmniHelpOptions]
; UseSearchHighlight = Yes (default, highlight search terms found)
; or No
UseSearchHighlight=Yes
; SearchHighlightStyle = style to use in span to highlight search
; terms found
SearchHighlightStyle=background-color:yellow;
```

The highlighting style consists of one or more CSS *property:value* pairs, each followed by a semicolon. Consult a CSS reference for possible styles.

10.11 Setting up CSH for OmniHelp

To provide entry points for CSH (context-sensitive help) calls from an application to an OmniHelp system, you can embed symbolic IDs as hypertext **newlink** markers in your FrameMaker files, and list alias prefixes in the configuration file. OmniHelp does not use numeric values for CSH, so you do not need a map file. Actually opening an OmniHelp topic file from an application might require a redirect page or a visit to the Windows Registry.

In this section:

- §10.11.1 [Specifying alias prefixes for OmniHelp CSH calls](#) on page 364
- §10.11.2 [Referencing OmniHelp topic IDs from an application](#) on page 365
- §10.11.3 [Using redirect pages for OmniHelp CSH calls](#) on page 365
- §10.11.4 [Executing browser commands for OmniHelp CSH calls](#) on page 366

See also:

- §7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239
- §34.1.2 [Using markers to add links and instructions](#) on page 935

10.11.1 Specifying alias prefixes for OmniHelp CSH calls

Mif2Go produces CSH alias entries from every **newlink** marker in your FrameMaker document whose content starts with one of the prefixes you specify. If you do not specify any prefixes, all **newlink** markers become aliases.

To specify one or more alias prefixes for CSH calls to OmniHelp topics:

```
[OmniHelpOptions]
; AliasPrefix = all prefixes wanted in alias file, comma or space
; delimited; if omitted, all newlinks are included
; NOTE: wildcards do not work in prefixes
AliasPrefix = prefix1, prefix2, ...
```

For example:

```
[OmniHelpOptions]
AliasPrefix = HIDC_, IDH_
```

With this setting, the alias file would include the content of every **newlink** marker in your document whose text starts with `HIDC_` or `IDH_`.

Mif2Go always creates the alias file, needed even if empty.

10.11.2 Referencing OmniHelp topic IDs from an application

To specify OmniHelp topic IDs to the browser, an application would make an **exec** file call with a file parameter that looks like one of the following:

```
_myproj.htm#file.htm           to get to a specific HTML file in myproj
_myproj.htm#IDH_contextID      to get to the file containing IDH_contextID
```

where *myproj* is a concatenation of the values specified in configuration section [OmniHelpOptions] for keywords `OHProjFilePrefix`, `ProjectName`, and `OHProjFileSuffix` (the underscore is the default value for `OHProjFilePrefix`). See §10.3 [Setting up an OmniHelp project](#) on page 345.

The second form (using `IDH_contextID`) is preferable, because it works even if the topic file name changes. Also, the second form provides a way for a WinHelp system to link to a specific topic in an OmniHelp system. The WinHelp project would not have to be recompiled if file names changed in the OmniHelp project.

10.11.3 Using redirect pages for OmniHelp CSH calls

If your application has trouble passing a topic-specific URL to the operating system (and then to the default browser), try creating a redirect page for each CSH target topic. A redirect page has content like this:

```
<!DOCTYPE html PUBLIC
"-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html lang="en">
<head><title>Topic title</title>
<meta http-equiv="refresh"
content="1;url=file:///path/to/_myproj.htm#IDH_contextID">
</head>
<body></body></html>
```

In this example, `IDH_contextID` is the content of a **newlink** marker in the target topic. You need one little file like this for every CSH entry point. This is not necessarily a bad thing; redirect files allow you to use a constant set of names in the calling program, even if the names change in the Help. Notice the *three* forward slashes in the file reference:

```
url=file:///path/to/_myproj.htm#IDH_contextID
```

For example:

```
url=file:///G:/Omnisys/UG/OH/Done/ugmif2go.htm#tablist
```

If you do not know the absolute path on the system where OmniHelp will be deployed, but you are able to place redirect files in the same directory as the OmniHelp output files (and invoke OmniHelp from that directory), you could use the following for the `url` value:

```
url=file:_myproj.htm#IDH_contextID
```

Relative paths would work like this:

```
url=file:./to/_myproj.htm#IDH_contextID
```

You could use a file name (*file.htm*) after the hash mark, with the same result; see §10.11.2 [Referencing OmniHelp topic IDs from an application](#) on page 365.

10.11.4 Executing browser commands for OmniHelp CSH calls

If the application that calls your OmniHelp project can execute system commands, the developer can have the application access the Windows Registry for the required browser command syntax, and use that command to open an OmniHelp topic file. With this method, you do not need redirect pages (see §10.11.3 [Using redirect pages for OmniHelp CSH calls](#) on page 365).

To see what is involved, check the Windows Registry for the correct browser command syntax (**Start > Run > regedit**):

1. Find the registered name of the default browser. Go to the following key:

```
HKEY_CLASSES_ROOT\.htm
```

and look at the first (Default) entry in the **Data** column. For example, for Firefox the default browser name is FirefoxHTML.

2. Find the exact command syntax for the default browser. Go to the following key:

```
HKEY_CLASSES_ROOT\DefaultBrowserName\shell\open\command
```

For example, for Firefox you would go to:

```
HKEY_CLASSES_ROOT\FirefoxHTML\shell\open\command
```

3. Look at the first (Default) entry in the **Data** column. For example, the command for Firefox might be:

```
C:\PROGRA~1\MOZILL~2\FIREFOX.EXE -url "%1"
```

In each call, the application should replace %1 in the browser command with the following type of file reference:

```
file:///path/to/_myproj.htm#IDH_contextID
```

where IDH_contextID is the content of a **newlink** marker in FrameMaker. Notice the *three* forward slashes in the file reference. This syntax should open the correct OmniHelp topic file.

10.12 Merging OmniHelp projects

An OmniHelp project can include links to one or more other OmniHelp projects that are located in other directories.

In this section:

§10.12.1 [Understanding the OmniHelp merge process](#) on page 366

§10.12.2 [Listing and mapping OmniHelp subprojects](#) on page 367

§10.12.3 [Providing TOC placeholders for OmniHelp subprojects](#) on page 368

§10.12.4 [Deciding when to merge OmniHelp subprojects](#) on page 369

See also:

§7.11 [Setting up a dynamic modular Help system](#) on page 241

10.12.1 Understanding the OmniHelp merge process

To merge files in one OmniHelp project with files in other OmniHelp projects, you must designate one of the projects to be the main project; the others are subprojects. The projects to be linked are merged at run time, when a user loads a project into a browser, or chooses a link that has a destination in another project. The merge process seamlessly integrates the contents of all *subproj.oh** files from each subproject into those of the main project; the result is just as if main and subprojects had always been one project.

<i>Project names must be unique</i>	Each OmniHelp project involved in a merge must have a unique project name. You must provide instructions in the main-project configuration file for merging subproject files when OmniHelp is loaded into a browser.
<i>Subprojects can be nested</i>	The merge process includes each subproject's merge data; as a result, other subprojects specified for merging into a given subproject are also integrated into the main project, allowing any degree of nesting of subprojects. However, OmniHelp does not support circular merging, where one subproject tries to merge another that has already been merged, or tries to merge the main project. Such anomalous merge attempts are ignored.
<i>Main project must know about subprojects</i>	The main project (and any subproject that includes other subprojects) must be aware of all existing and potential subprojects at the next level down, whether or not those subprojects are actually present when the main project is loaded into a browser. The name and title of each subproject must be listed in the main project's configuration file (see §10.12.2 Listing and mapping OmniHelp subprojects on page 367), and each subproject must have an entry in the including project's table of contents (see §10.12.3 Providing TOC placeholders for OmniHelp subprojects on page 368).

10.12.2 Listing and mapping OmniHelp subprojects

To designate subprojects to be merged, list the subproject paths and titles. For example:

```
[HelpMerge]
; Subproject name and path = Title of subproject
..\api\LibRef = API Library Reference
```

Each [HelpMerge] entry specifies a subproject base file name, including a relative path if the subproject is in a different directory, and the title to be displayed for that subproject in the main-project contents. The relative path must be correct at run time.

If your main-project FrameMaker document includes links to FrameMaker files that are part of a subproject listed in [HelpMerge], specify how those linked-to FrameMaker files are mapped to their subprojects. For example:

```
[OHMergeFiles]
; Mapping of Frame files which are linked to by files in the current
; project, but are themselves part of another OH project that is
; listed in [HelpMerge], to the name of that project.
; Omit all file extensions.
; file path = subproject path (no prefix, no extension)
D:/Library/AppxB=../api/LibRef
```

It is best to include a path, because you could have several files with the same name (such as Glossary) in several different books from which you generate different OmniHelp projects; without file paths, you would have no way to differentiate them.

To handle several possible paths to the same file, you can add a line for each path. For example:

```
[OHMergeFiles]
ChapA=BookA
ChapB=BookB
.../GroupB/ChapB=BookB
G:/test/GroupB/ChapB=BookB
```

Although you can use either forward slashes or backslashes in paths, forward slashes are preferred. FrameMaker stores cross-reference paths with forward slashes, and Mif2Go uses those cross-reference paths to find the referenced files.

10.12.3 Providing TOC placeholders for OmniHelp subprojects

Place a **HelpMerge** marker in your main-project FrameMaker document for each subproject listed in the [HelpMerge] section, to:

- show where the subproject TOC should be merged into the main project TOC
- specify a contents level for the top TOC entry for the subproject.

Insert the **HelpMerge** marker between two main-project topics, in either of the following places:

- at the start of the main-project topic that should follow the subproject in the TOC, before any text
- at the end of the main-project topic that should precede the subproject in the TOC, after all text, in an otherwise empty paragraph.

Do not place the **HelpMerge** marker at the very beginning of the main project, and do not include duplicate **HelpMerge** markers for the same subproject.

The content of the **HelpMerge** marker consists of a single-digit contents level number (with respect to the main project TOC) for the top TOC entry of the subproject, followed by a space, followed by the path to the subproject. For example:

```
1 ../api/LibRef
```

*Add subprojects
before the fact*

To include merge points for future subprojects that are not yet available, so that the main project does not even need to know whether they exist, at the end of the TOC add extra merge points with dummy subproject names. If you specify load-time merging:

```
[OmniHelpOptions]
MergeFirst=Yes
```

any subprojects that are present will be integrated, and any merge point for which a subproject is not present will be removed from the TOC.

*Add subprojects
after the fact*

To provide the marker content after the fact, for a subproject that has already been built or was created without using **Mif2Go**, insert an entry in the `*_ohc.js` file for the master project, in the position where you want the subproject entry to appear in the master-project contents. The entry must look like this:

```
[n,"title","*name"],
```

where the components are as follows:

<code>n</code>	Contents level for the subproject entry in the master-project contents
<code>title</code>	Title of the subproject
<code>name</code>	Project name of the subproject

The last three items in the following example identify subprojects: that are in a directory different from the parent directory, so a relative path is prefixed to the project name:

```
var tocItems = [
  [1,"Server","aa998290.htm#Xaa998290"],
  [2,"Feature 1","aa998295.htm#Xaa998295"],
  [2,"Feature 2","aa998300.htm#Xaa998300"],
  [1,"Connectors","aa998313.htm#Xaa998313"],
  [2,"Connector A","*ConnA/ConnA"],
  [2,"Connector B","*ConnB/ConnB"],
  [2,"Connector C","*ConnC/ConnC"]]
```

You would also need an item in `_ohx.js` like this:

```
var mergeProjects = [
  ["ConnA/ConnA",0,0,4,[]],
```

```
[ "ConnB/ConnB", 0, 0, 5, [ ] ],
[ "ConnC/ConnC", 0, 0, 6, [ ] ]
```

where the 4, 5, 6 are the (zero-based) numbers of the TOC items. This example is for a set-up in which each secondary item is in a subdirectory that has the same name as the project.

See also:

§7.4.4 [Setting contents levels for HTML-based Help](#) on page 210

§10.12.2 [Listing and mapping OmniHelp subprojects](#) on page 367

§29.2 [Adding custom marker types](#) on page 832

10.12.4 Deciding when to merge OmniHelp subprojects

You can specify whether to merge all subprojects when a browser first loads the OmniHelp main project, or to merge a given subproject only when a user chooses a link to that subproject. The default is to merge only on demand.

To merge all subprojects when the main project is first loaded:

```
[OmniHelpOptions]
; MergeFirst = No (default, merge subprojects only when they are
; called on), or Yes (do all merges during initial load,
; takes longer to start up)
MergeFirst=Yes
```

*Merge at load
time*

When MergeFirst=Yes, at browser load time the contents and index entries of all subprojects that meet the following criteria are merged with those of the main project:

- The subproject name is listed under [HelpMerge] in the main project configuration file; see §10.12.2 [Listing and mapping OmniHelp subprojects](#) on page 367.
- The main project's table of contents includes a merge point for the subproject; see §10.12.3 [Providing TOC placeholders for OmniHelp subprojects](#) on page 368.
- The subproject files are actually present.

Merge-point entries are quietly removed from the main project table of contents for any subproject whose files are not present at load time. Merging subprojects at load time makes the browser load process significantly slower than just loading the main project. You would *not* want this option if you are using OmniHelp for context-sensitive help (see §10.11 [Setting up CSH for OmniHelp](#) on page 364). However, this is the way to go if you distribute non-CSH OmniHelp systems that do not always include all subprojects.

*Merge on
demand*

When MergeFirst=No (the default), at browser load time all subproject merge-point entries are included in the main project table of contents, whether or not the subproject files are actually present. When a user clicks a subproject entry, if the subproject files are present, the subproject contents and index entries are merged with the main project contents and index entries. However, if a user clicks an entry for a missing subproject, that entry disappears from the main project table of contents.

10.13 Assembling OmniHelp files for viewing

By default, **Mif2Go** copies all files with the following extensions from the project directory to the wrap directory (see §35.2 [Activating and logging production of deliverables](#) on page 956):

```
*.htm *.css *.js *.gif *.jpg *.png
```

You can change the list of files to be copied; see §35.6 [Assembling files for distribution](#) on page 961. **Mif2Go** automatically copies the necessary OmniHelp viewer files from the viewer-control directory to the wrap directory, according to the value of OHViewPath:

```
[OmniHelpOptions]
; OHViewPath = path to dir containing the OH viewer files
```

See §10.2.2 [Making OmniHelp viewer control files available](#) on page 343. By default, OHViewPath references the OmniHelp viewer files in one of the following directories:

```
%OMSYSHOME%\common\system\omnihelp\ohvhtm (for HTML output)
%OMSYSHOME%\common\system\omnihelp\ohvxml (for XHTML output)
```

If you put the viewer-control files somewhere else, you must specify the path (preferably absolute) to that location as the value of OHViewPath. Do not place the viewer-control files under the wrap directory. **Mif2Go** copies the files listed in [Table 10-5](#) from the directory designated by OHViewPath to the directory designated by WrapPath, if specified, otherwise to the project directory.

Table 10-5 OmniHelp viewer files copied from OHViewPath to WrapPath

Start-up file type	OmniHelp viewer files copied to wrap directory by default
HTML	oh*.*
XHTML	ox*.htm ox*.js oh*.css oh*.js

The start-up file type (HTML or XHTML) determines which set of files will be copied; see §10.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 342.

To have **Mif2Go** copy additional files to the wrap directory:

```
[OmniHelpOptions]
; OHVFiles = list of files to copy from OHViewPath (the viewer files).
OHVFiles = oh*.* some\other\files yet\more\files ...
```

The file specifications you assign to OHVFiles must be separated by spaces, and no spaces are allowed within a file specification. You can use wildcards in file specifications, and include absolute or relative paths to indicate where viewer files should be copied from. Relative paths are relative to the wrap directory.

The files you list for OHVFiles will be copied *in addition to* the files listed in [Table 10-5](#). If you are not adding any special files of your own, there is no need to include a setting for OHVFiles. When you do not provide a setting for OHVFiles, the default value is based on the setting for OHProjFileXhtml; see §10.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 342.

See also:

§7.2.4 [Compiling and distributing Help systems](#) on page 204

§35.6 [Assembling files for distribution](#) on page 961

10.14 Deploying OmniHelp

When users launch an OmniHelp system, what happens depends on which browser they are using, and how they tell the browser to load OmniHelp. Most browsers will refuse to open HTML files on a non-local drive that are called using the file protocol. If the files are not on a local drive, they must be served by an HTTP server and called using the http protocol.

In this section:

§10.14.1 [Starting with the default topic or a specified topic](#) on page 371

§10.14.2 [Restarting where you left off](#) on page 371

§10.14.3 [Coping with browser quirks](#) on page 371

10.14.1 Starting with the default topic or a specified topic

To launch OmniHelp, you can specify any of the following in the locator field of the browser, or in a link in some other file:

<code>_myproj.htm</code>	Loads everything else, starting with the default topic file.
<code>_myproj.htm#filename.htm</code>	If <code>filename.htm</code> is a topic file in the OmniHelp project, the browser shows <code>filename.htm</code> first.
<code>_myproj.htm#name</code>	If there is no dot in <code>name</code> , the browser looks up <code>name</code> in OmniHelp data file <code>myproj_oha.js</code> , and loads the appropriate file; see §10.11 Setting up CSH for OmniHelp on page 364.

In these examples `myproj` is a concatenation of the values specified in configuration section `[OmniHelpOptions]` for keywords `OHProjFilePrefix`, `ProjectName`, and `OHProjFileSuffix`; the underscore is the default value for `OHProjFilePrefix`. See §10.3 [Setting up an OmniHelp project](#) on page 345.

10.14.2 Restarting where you left off

Once an OmniHelp project is loaded, the browser constantly stores the current state in cookies that last one year. If you exit OmniHelp, the next time you load it, you are back where you were before. To get to the beginning (the default topic file) instead, click **Start**.

10.14.3 Coping with browser quirks

If you click the **Reload** button on your browser with your OmniHelp system loaded, CSS style sheets might not reload, so the resulting page might appear unformatted. Browsers do not retain the original URL internally, and if you try to restart from the later-stage OmniHelp file the browser does recall, you miss loading several necessary JavaScript files. That is why **Mif2Go** provides a **Start** button. Use the OmniHelp **Start** button instead of the browser **Reload** button.

Likewise, never use the browser **Back** button; always use the OmniHelp **Back** button instead. When you load OmniHelp in Internet Explorer, this is not a problem, because Internet Explorer loads in its own window that does not have these problematic browser controls. Although you can do the same in Firefox, thanks to its “security” features, this works only when you are loading from the Web, not locally.

The most commonly used browsers on Windows each seem to have a different issue with displaying OmniHelp files:

- [Internet Explorer issues](#)
- [Firefox issues](#)
- [Chrome issues](#)
- [Opera issues](#)
- [Safari issues](#)
- [Netscape issues](#)

- Internet Explorer issues* When you open an OmniHelp file in Internet Explorer, even if you have specified that the existing window should be closed (see §10.5.1 [Configuring OmniHelp window usage and frameset dimensions](#) on page 352), you get a confirmation dialog:
- The Web page you are viewing is trying to close the window.
Do you want to close this window?*
- This is an Internet Explorer “security feature” that cannot be turned off. To avoid the confirmation dialog, your only real choice is to open OmniHelp in the existing window, with all the browser chrome on top. Or open in the new window, but leave the starting window open too, which looks like a mistake but is harmless.
- Firefox issues* Firefox does not open a new window when you launch a local OmniHelp system by double-clicking `_myproj.htm`, unless you also set the following option in Firefox. On the main Firefox menu, choose:
- Tools > Options... > Tabs > Open links from other applications in:**
- and check **a new window**. Unfortunately, all the chrome comes along with the new window.
- For OmniHelp systems viewed on the Web, unless you have pop-up windows blocked, Firefox should open OmniHelp in a new window, without chrome. If you do have pop-up windows blocked, you can unblock them selectively; on the main Firefox menu, choose:
- Tools > Options... > General > Block Popup Windows > Allowed Sites**
- and add the Web address where your OmniHelp system is located.
- If you click **Reload** to refresh OmniHelp in Firefox, the left navigation pane might lose its CSS rendering. The workaround is to close the OmniHelp tab, then reopen OmniHelp from a Firefox bookmark that references `_myproj.htm` (see §10.14.1 [Starting with the default topic or a specified topic](#) on page 371).
- Chrome issues* When you attempt to access Help files located in your local file system, OmniHelp (and all other forms of Web Help we know about) will not work in Google Chrome, unless you start Chrome with this special command-line switch:
- `--allow-file-access-from-files`
- This option allows locally hosted Web Help systems to open in Chrome. Otherwise, Chrome does not allow local files to access the JavaScript scope of the parent frame/window. Because of security risks, users should start Chrome with this option only to view trusted local Web Help systems.
- See Peter Grainge’s discussion of this issue, in Snippet 130:
<http://www.grainge.org/pages/snippets/snippets.htm>
- Opera issues* On some systems, Opera works as expected with OmniHelp. On other systems, Opera might not display the left navigation pane. On still other systems, refresh eliminates the content of the contents, the index, and the search frame.
- Safari issues* On an iPad, Safari does not seem to respect frame size settings. Instead the frame adjusts to the width of its widest contents.
- Netscape issues* Later versions of Netscape Navigator might refuse to open OmniHelp files if you have suppressed pop-ups; on the Navigator **Edit** menu, look at **Preferences... > Privacy & Security > Pop-up Windows**. Also, later versions of Netscape Navigator might ignore CSS for OmniHelp files viewed over the Web. Local OmniHelp files, with local CSS, are displayed properly.

11 Generating JavaHelp or Oracle Help

This section addresses issues that are specific to generating JavaHelp and Oracle Help for Java. HTML settings described in section 13 and sections 18 through 34 apply also. Topics include:

- §11.1 [Deciding which Java Help system to use](#) on page 373
- §11.2 [Obtaining tools for a Java-based Help system](#) on page 373
- §11.3 [Setting up a JavaHelp or Oracle Help project](#) on page 374
- §11.4 [Generating contents and index](#) on page 385
- §11.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 387
- §11.6 [Creating and viewing a Java Archive \(JAR\) file](#) on page 390
- §11.7 [Converting a glossary to JavaHelp 2](#) on page 392
- §11.8 [Defining windows for JavaHelp or Oracle Help](#) on page 393
- §11.9 [Linking to destinations within topics](#) on page 399
- §11.10 [Creating ALinks for Oracle Help](#) on page 399
- §11.11 [Merging JavaHelp or Oracle Help systems](#) on page 400
- §11.12 [Setting up CSH for JavaHelp or Oracle Help](#) on page 401

See also:

- §7 [Producing on-line Help](#) on page 199

11.1 Deciding which Java Help system to use

JavaHelp and Oracle Help for Java offer true platform independence, provided a Java Virtual Machine (JVM) is available for each platform you support. However, JavaHelp is no longer supported, so it is not recommended.

About JavaHelp

JavaHelp 2.0 provides features such as a “favorites” list, support for a glossary, and support for secondary windows. You can download the *JavaHelp System User’s Guide* in PDF format here:

<http://download.java.net/javadesktop/javahelp/>

Earlier versions of JavaHelp are no longer available, and current versions are no longer supported.

Mif2Go produces HTML 3.2 code for JavaHelp. The HTML 3.2 code works with the W3C validator to validate JavaHelp topic files, with one exception: JavaHelp cannot abide single quotes in <meta> tags in the <head> element, so **Mif2Go** omits them.

About Oracle Help for Java

Oracle Help for Java has all the capabilities of JavaHelp; can use the same files; and supports some nice extensions, such as ALinks. Information is available from the Oracle Technology Network:

<http://www.oracle.com/technetwork/topics/index-083946.html>

You must register to access the Oracle site, but registration is free.

11.2 Obtaining tools for a Java-based Help system

You will need Java Standard Edition (Java SE): version 2 or later for JavaHelp, version 5 or later for Oracle Help. Download the Java SE from this site:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Note: Install Java SE in a directory with no spaces in the path name; otherwise you will have to modify some of the scripts that accompany the Help tools, to enclose the path name in double quotes.

Download Oracle Help for Java from this site:

<http://www.oracle.com/technetwork/topics/ohj50ext-089966.html>

Download JavaHelp from this site:

<http://download.java.net/javadesktop/javahelp/>

Note: Install JavaHelp in a directory with no spaces in the path name.

Supposedly, eventually JavaHelp downloads should be available here:

<http://java.net/projects/javahelp/>

However, it does not appear that this site is maintained.

For both Help systems the software is free, and can be redistributed.

11.3 Setting up a JavaHelp or Oracle Help project

In this section:

- §11.3.1 [Creating a JavaHelp or Oracle Help for Java project](#) on page 374
- §11.3.2 [Choosing set-up options for a JavaHelp or Oracle Help project](#) on page 375
- §11.3.3 [Deciding where to locate configuration settings](#) on page 376
- §11.3.4 [Specifying output options for JavaHelp](#) on page 376
- §11.3.5 [Establishing a JavaHelp environment](#) on page 377
- §11.3.6 [Establishing an Oracle Help environment](#) on page 377
- §11.3.7 [Creating a directory structure for JavaHelp / Oracle Help](#) on page 378
- §11.3.8 [Configuring the helpset file](#) on page 382
- §11.3.9 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 384
- §11.3.10 [Compiling JavaHelp with Helen](#) on page 384

See also:

- §7.2.1 [Checking automatic Help topic assignments](#) on page 203

11.3.1 Creating a JavaHelp or Oracle Help for Java project

To create a JavaHelp or Oracle Help project:

1. Create a project directory for output files, separate from the directory where your FrameMaker document is located.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
3. Name your JavaHelp or Oracle Help project, and browse to the project directory you created in [Step 1](#).
4. Choose output type JavaHelp or output type OracleHelp and click **OK**.
5. Check options in the *Set Up Java Help Project* dialog (see §11.3.2 [Choosing set-up options for a JavaHelp or Oracle Help project](#) on page 375).
6. Use a text editor to edit the resulting `_m2javahelp.ini` or `_m2oraclehelp.ini` configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).

7. **Important:** To make the configuration fit your usage of headings to start topics, pay special attention to sections [HelpContentsLevels] (see §7.2.1 [Checking automatic Help topic assignments](#) on page 203) and [HTMLParaStyles] (see §18.2 [Splitting files](#) on page 586).

11.3.2 Choosing set-up options for a JavaHelp or Oracle Help project

When you select JavaHelp or Oracle Help for Java as the output type for a new project, the *Set Up* dialog shown in [Figure 11-1](#) opens. [Table 11-1](#) shows the corresponding settings in the configuration file. *You must edit the configuration file to specify additional options.*

See also:

§3.4 [Choosing project set-up options](#) on page 79

§7 [Producing on-line Help](#) on page 199

§13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425

Figure 11-1 Set Up Java Help Project

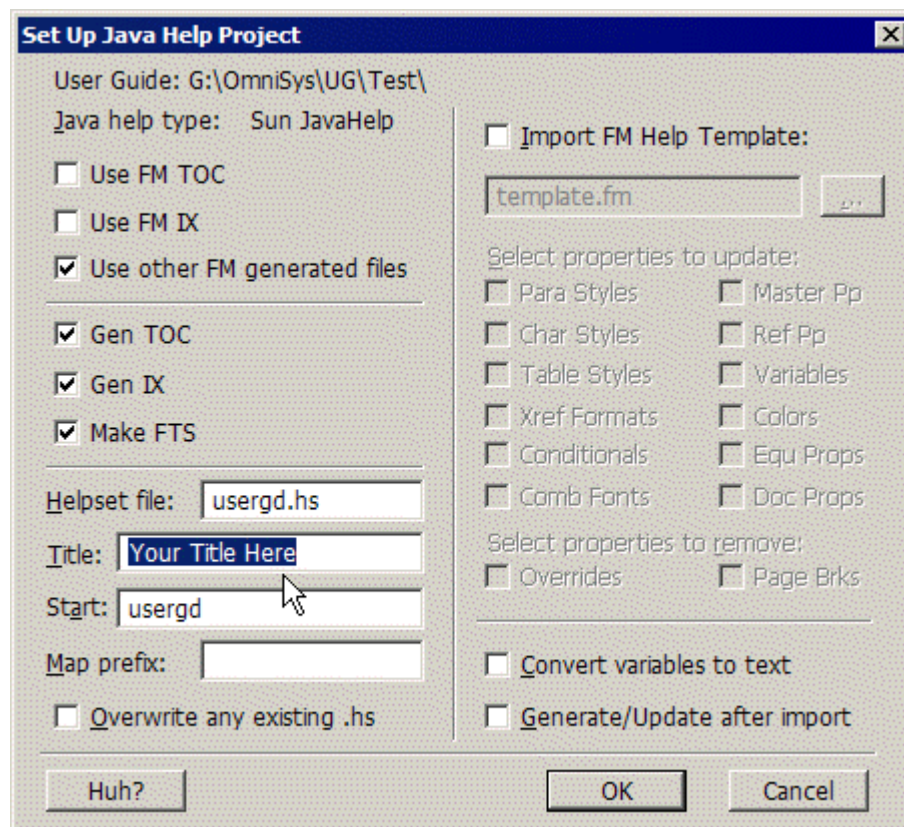


Table 11-1 JavaHelp set-up options and configuration settings

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Gen TOC	[JavaHelpOptions]	ListType=Contents	Both	7.3.4.1
Gen IX	[JavaHelpOptions]	ListType=Index	Both	7.3.4.1
FTS:	[JavaHelpOptions]	UseFTS=Yes	Omitted	11.5
	[JavaHelpOptions]	FTSCommand=path/to/FTS /program	Omitted	11.5
Jar with:	[Automation]	ArchiveCommand=path/to /JAR/program	Omitted	11.6

Table 11-1 JavaHelp set-up options and configuration settings (continued)

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Helpset file	[JavaHelpOptions]	HSFileName=MyDoc.hs	MyDoc.hs	11.3.8
Title	[JavaHelpOptions]	HelpSetTitle=My Title	Your Title Here	11.3.8
Start	[JavaHelpOptions]	DefaultTopic=Topic ID	MyDoc	11.3.8
Map prefix	[JavaHelpOptions]	MapFilePrefix=html/	Omitted	11.3.7.4
Overwrite any .hs	[JavaHelpOptions]	WriteHelpSetFile=Yes	No	11.3.8

11.3.3 Deciding where to locate configuration settings

When you set up a JavaHelp or Oracle Help project from within FrameMaker, if configuration file `_m2javahelp.ini` (or `_m2oraclehelp.ini`) is not already present in your project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

Which configuration file?

To configure output, add settings to one of the following files, depending on the desired scope of each setting.

For *JavaHelp* output:

<u>Scope</u>	<u>Configuration file</u>	<u>Location</u>
Current project only	<code>_m2javahelp.ini</code>	Current project directory
All JavaHelp projects	<code>local_m2javahelp_config.ini</code>	<code>%omsyshome%\m2g\local\config\projects</code>

For *Oracle Help* output:

<u>Scope</u>	<u>Configuration file</u>	<u>Location</u>
Current project only	<code>_m2oraclehelp.ini</code>	Current project directory
All Oracle Help projects	<code>local_m2oraclehelp_config.ini</code>	<code>%omsyshome%\m2g\local\config\projects</code>

See §30.5 [Deciding which configuration file to edit](#) on page 856.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §13 [Converting to HTML/XHTML](#) on page 423
- §18 [Splitting and extracting files](#) on page 585
- §21 [Mapping text formats to HTML/XML](#) on page 645
- §23 [Including graphics in HTML](#) on page 703
- §24 [Converting tables to HTML](#) on page 727

11.3.4 Specifying output options for JavaHelp

By default, **Mif2Go** produces JavaHelp 2 output. To generate JavaHelp 1 instead:

```
[JavaHelpOptions]
; JHVersion2 = Yes (default) or No (limit features used to Version 1)
JHVersion2 = No
```

If you do choose JavaHelp 1, be aware that several executable files in the JavaHelp distribution that used to be `.exe` files were changed to `.jar` files in JavaHelp version

1.1.3. For example, with version 1.1.3 you use `jhindexer.jar` and `hsvviewer.jar`. You must run both `JHIndexer` and `hsvviewer` from the command line, and you will need environment variable `JAVAHELP_HOME` to run them at all.

See also:

§11.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 387

§11.6 [Creating and viewing a Java Archive \(JAR\) file](#) on page 390

11.3.5 Establishing a JavaHelp environment

To use JavaHelp, you must have both JavaHelp and the Java Runtime Environment (JRE) installed on your system, and you must set some environment variables. If you plan to create `.jar` files, you will also need `jar.exe` from the Java Software Development Kit (JDK).

JavaHelp 2.0 requires Java Standard Edition (Java SE). Both Java SE and JavaHelp are available for download; see §11.2 [Obtaining tools for a Java-based Help system](#) on page 373.

*Environment
variables*

You must create Windows environment variables `JAVA_HOME` and `JHHOME`, if they are not already defined on your system; for example, on Windows 2000 or Windows XP:

Control Panel > System > Advanced > Environment Variables

These environment variables are defined as follows:

```
JAVA_HOME    path\to\JavaSE\executables
JHHOME       path\to\JavaHelp\executables
```

For example:

```
JAVA_HOME=C:\Java\j2re1.4.2_03\bin
JHHOME=C:\JH\jh20\javahelp\bin
```

JavaHelp viewer

To check the results after **Mif2Go** generates JavaHelp files, you can use the JavaHelp viewer included in the JavaHelp installation: `hsvviewer.jar`, located in the `demos\bin` directory. The *JavaHelp System User's Guide* shows how to set up a shortcut to the viewer.

11.3.6 Establishing an Oracle Help environment

To use Oracle Help for Java, you must have both Oracle Help and a Java Virtual Machine (JVM) installed on your system. Oracle Help version 5.0 requires Java SE version 5.0 or a later version. If you plan to create `.jar` files, you will also need `jar.exe` from the Java Developer's Kit (JDK).

Oracle Help for Java is available for download from the Oracle Technology Network; see §11.2 [Obtaining tools for a Java-based Help system](#) on page 373.

*Environment
variables*

Edit Windows System environment variable `CLASSPATH` (or create `CLASSPATH` if it is not already defined on your system):

Control Panel > System > Advanced > Environment Variables

What you add to `CLASSPATH` depends on which version of Oracle Help you are using; the dependencies and file names changed between versions 4 and 5.

If you are using Oracle Help version 4, append to `CLASSPATH` the following paths, separating each path from the next with a semicolon:

```
where\you\installed\ohj\
where\you\installed\ohj\help4.jar
where\you\installed\ohj\help4-demo.jar
```



```
where\you\installed\ohj\help4-indexer.jar
where\you\installed\ohj\ohj-jewt.jar
where\you\installed\ohj\oracle_ice.jar
```

For example (all on one line, of course):

```
CLASSPATH=D:\ohelp\help4-indexer.jar;D:\ohelp\help4-demo.jar;D:\ohelp\
help4.jar;D:\ohelp\ohj-jewt.jar;D:\ohelp\;D:\ohelp\oracle_ice.jar;D:\o
help\help4-indexer.jar
```

If you are using Oracle Help version 5, append to CLASSPATH the following paths, separating each path from the next with a semicolon:

```
where\you\installed\ohj\
where\you\installed\ohj\ohj.jar
where\you\installed\ohj\help-share.jar
where\you\installed\ohj\share.jar
where\you\installed\ohj\help-demo.jar
where\you\installed\ohj\help-indexer.jar
where\you\installed\ohj\jewt.jar
where\you\installed\ohj\oracle_ice.jar
```

For example (all on one line, of course):

```
CLASSPATH=g:\ohj5;g\ohj5\ohj.jar;g:\ohj5\help-share.jar;g:\ohj5\oracle
_ice.jar;g:\ohj5\jewt.jar;g:\ohj5\share.jar;g:\ohj5\help-indexer.jar
```

Oracle Help
viewer

Given these settings for CLASSPATH, if you have also established a path to `java.exe` in a current JRE (see §11.3.5 [Establishing a JavaHelp environment](#) on page 377), to view the results of generating Oracle Help you should be able to use a `.bat` file with commands like the following:

```
cd where\you\installed\ohj
java oracle.help.demo.ChoiceDemo "%path\to\MyOutput\help\MyDoc.hs"
```

In practice, for Oracle Help 5, we find that setting the CLASSPATH environment variable is not sufficient; you must still supply the same dependencies to the `java` command as an argument to `-classpath`. For example:

```
cd G:\OHJ5
REM The following java command must be all on one line:
java -classpath
"ohj.jar;help-share.jar;oracle_ice.jar;jewt.jar;share.jar;help-demo.jar
r" oracle.help.demo.ChoiceDemo "G:\OmniSys\UG\ohj\help\ugmif2go.hs" %*
```

Your experience might be different.

11.3.7 Creating a directory structure for JavaHelp / Oracle Help

In this section:

- §11.3.7.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 378
- §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379
- §11.3.7.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 380
- §11.3.7.4 [Specifying a path for search-index links](#) on page 381
- §11.3.7.5 [Manually copying and deleting output files](#) on page 381

11.3.7.1 Understanding the JavaHelp / Oracle Help directory structure

If you plan to provide features such as full-text search, JavaHelp and Oracle Help require a more involved directory structure than just a **Mif2Go** project directory. You can create the directory structure, or let **Mif2Go** do it for you; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379. A typical directory structure looks like this,

with the top-level JavaHelp or Oracle Help directory as a subdirectory of the conversion project directory:

<i>MyDoc</i>	FrameMaker document files
<i>..MyOutput</i>	Mif2Go output files (normal Mif2Go project directory)
<i>Top JH or OHJ level starts here:</i>	
<i>....help</i>	Help files copied from <i>MyOutput</i> : .hs, .jhm, .xml
<i>.....graphics</i>	Image files copied from <i>MyOutput</i> (or another location)
<i>.....html</i>	.htm and .css files copied from <i>MyOutput</i>

After **Mif2Go** generates output, the helpset file (.hs) and navigational files (.jhm and .xml) are copied from the **Mif2Go** project directory to the help directory.

11.3.7.2 Letting Mif2Go set up the directory structure and copy files

To have **Mif2Go** set up the JavaHelp or Oracle Help directory structure for you, specify a path to the top-level directory. For example:

```
[Automation]
WrapAndShip = Yes
; WrapPath = for JavaHelp or Oracle Help, path to top-level dir,
; default is output dir
WrapPath = ./help
```

WrapPath can be an absolute path or a path relative to the project directory; the default value of WrapPath is the project directory itself.

*Directories are
created*

When you specify a value for WrapPath, **Mif2Go** creates the WrapPath directory if it is not already present, and also creates the two required subdirectories, if they are not already present.

To specify names for the subdirectories:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; HTMLSubdir = subdirectory of WrapPath for *.htm, *.css, and *.js
; files, default "html"
HTMLSubdir = html
; GraphSubdir = subdirectory of WrapPath for *.gif, *.jpg, and *.png
; files, default "graphics"
GraphSubdir = graphics
```

Unless you are creating a proprietary directory structure, just accept the default names.

The directory designated by HTMLSubdir is the default setting for MapFilePrefix, with “/” appended; see §11.3.7.4 [Specifying a path for search-index links](#) on page 381.

The directory designated by GraphSubdir is the default JavaHelp and Oracle Help setting for [Graphics]GraphPath, with “./” prepended; see §11.3.7.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 380.

*Directories can be
emptied before
copying*

To empty the subdirectories before copying:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; EmptyJavaHTMLSubdir = Yes (default, empty HTMLSubdir directory
; before copying) or No (leave HTML files in place)
EmptyJavaHTMLSubdir = Yes
; EmptyJavaGraphSubdir = No (default, leave graphics files in place)
; or Yes (empty GraphSubdir directory before copying)
EmptyJavaGraphSubdir = Yes
```

*Files are copied
from the project
directory*

When you specify a value for [Automation]WrapPath, **Mif2Go** automatically populates the directory structure. After generating HTML files and optionally creating a full-text search index, **Mif2Go** copies files that have the following extensions, from the project directory to the directory specified by WrapPath, or to the appropriate

subdirectory. For example, with WrapPath=. /help and default names for the subdirectories:

<u>Directory</u>	<u>File extensions</u>
.\help	*.xml *.hs *.jhm
.\help\html	*.htm *.css *.js
.\help\graphics	*.gif *.jpg *.png

Note: Files are automatically copied from the project directory only if you specify a value for WrapPath.

*List files to copy
to the top
directory*

To specify what files to copy to the top directory:

```
[JavaHelpOptions] or [OracleHelpOptions]
; JavaRootFiles = list of files to copy to WrapPath
JavaRootFiles = *.hs *.jhm *.xml
```

You can use JavaRootFiles to list files to be copied to the directory designated by WrapPath. The file specifications you assign to JavaRootFiles must be separated by spaces, and no spaces are allowed within a file specification. You can use wildcards in file specifications, and include absolute or relative paths to indicate where files should be copied from; the default is from the project directory. By default, the following files are copied:

```
*.hs *.jhm *.xml
```

Any file list you assign to JavaRootFiles overrides these defaults.

*Graphics can be
copied from a
different directory*

To have **Mif2Go** copy graphics files from a location other than the project directory:

```
[Automation]
WrapAndShip = Yes
CopyOriginalGraphics = Yes
```

When CopyOriginalGraphics=Yes, **Mif2Go** follows the file paths in your FrameMaker source to find the graphics files to copy to the directory specified by GraphSubdir.

See also:

§7.2.4 [Compiling and distributing Help systems](#) on page 204.

§35 [Producing deliverable results](#) on page 955

11.3.7.3 Locating graphics files for JavaHelp and Oracle Help

To view images in a generated JavaHelp system, if you are using a typical directory structure, image files must be in a subdirectory of the helpset directory such as help\graphics, and HTML files that reference the images must be in a parallel subdirectory, such as help\html. (See §11.3.7.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 378).

When you have finished generating a JavaHelp or Oracle Help system, both of the following must be true:

- all graphics referenced from HTML topic files are in one subdirectory of the helpset directory; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379
- all references to graphics specify a relative path from the helpset directory to the graphics subdirectory.

For example, to specify a relative path from directory help (where the helpset is located) to subdirectory help\graphics (where graphics are located):

```
[Graphics]
GraphPath = ../graphics/
```

For JavaHelp and Oracle Help (only), the directory designated by GraphSubdir (see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379) is the default setting for [Graphics]GraphPath, with “../” prepended.

Note: For JavaHelp and Oracle Help, forward slashes are required in path names you assign to keywords in the configuration file; see §4.4 [Understanding the rules for configuration settings](#) on page 102.

Mif2Go uses the value of GraphPath for the src attribute of tags. See §31.3.1.1 [Specifying graphics location for HTML](#) on page 887.

To specify the location of images assigned to specific JavaHelp 2 windows, see §11.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 394.

11.3.7.4 Specifying a path for search-index links

To create a search index, URLs in the JavaHelp map file (.jhm) must point to files in a subdirectory of the directory where the helpset file is located; the default subdirectory is help\html. See §11.3.7.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 378.

Note: Oracle Help uses a map file only for ALinks and for CSH links; if your Oracle Help project does not include either of those features, there is no map file.

To provide a prefix that points map-file URLs to the correct directory:

```
[JavaHelpOptions] or [OracleHelpOptions]
; MapFilePrefix = prefix to insert at start of map file URLs
MapFilePrefix = html/
```

The directory designated by HTMLSubdir (see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379) is the default setting for MapFilePrefix, with “/” added. Use a forward slash, not a backslash, at the end of the prefix; URLs require forward slashes.

MapFilePrefix fixes URLs in the map file, but does not actually move any of the referenced files; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

11.3.7.5 Manually copying and deleting output files

To get rid of any orphaned HTML files left over from a previous conversion run, for a stand-alone project (no links to other projects) it is best to delete all HTML output files before each full conversion, then copy new HTML output files to the appropriate compilation directory after conversion. For large projects, deleting and then recreating the help\html subdirectory is noticeably faster than deleting HTML files one by one.

*Do not delete until
after all
conversions*

If HTML output files contain cross references to another project (as in a merge situation), those cross references would be broken if you were to delete HTML files from the project directory, because **Mif2Go** would not be able to update the missing files. In that situation, update all projects that need updating before you copy HTML files from the project directory to the compilation directory; and delete HTML files from the project directory only when you start a full conversion of every project involved.

11.3.8 Configuring the helpset file

In this section:

- §11.3.8.1 [Specifying helpset file name and title](#) on page 382
- §11.3.8.2 [Specifying a default starting topic for the helpset](#) on page 382
- §11.3.8.3 [Deciding whether to rewrite the helpset file](#) on page 383
- §11.3.8.4 [Providing a “favorites” option for JavaHelp 2](#) on page 383
- §11.3.8.5 [Adding custom helpset sections for JavaHelp 2](#) on page 383
- §11.3.8.6 [Requiring full paths in the helpset file](#) on page 383

11.3.8.1 Specifying helpset file name and title

To specify a helpset file name for JavaHelp or Oracle Help:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; HSFileName = name for JavaHelp HelpSet file used in archive
HSFileName = myproj.hs
```

The default helpset file name is the name of your FrameMaker book or document file.

To have **Mif2Go** copy the helpset file to another directory after generating output files, specify the following:

```
[Automation]
; WrapPath = for JavaHelp or Oracle Help, path to top-level dir,
; default is output dir
WrapPath = ./help
```

See §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

To specify a helpset title:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; HelpSetTitle = title in HelpSet file, default filename or bookname
HelpSetTitle = Title of My Project
```

Oracle Help for Java does not support entities in the title. Do not include special characters such as &, <, >, or " in the title of the helpset file.

11.3.8.2 Specifying a default starting topic for the helpset

To specify the helpset starting topic:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; DefaultTopic = starting topic ID (not file name)
;DefaultTopic =
```

This setting specifies an identifier for the first topic to display. This is a JavaHelp-specific target name rather than a file name. Typically it is the ObjectID of the first heading in the file; for example, Xaa123456. The default-topic identifier appears in a helpset file entry such as the following:

```
<homeID>Xaa123456</homeID>
```

The helpset entry identifies the map-file URL that points to the topic file; for example:

```
<mapID target="Xaa123456" url="html/ugmif2go.htm" />
```

If you do not specify a value for `DefaultTopic`, **Mif2Go** tries to autodetect the target name; however, this works only if **Mif2Go** is rewriting the helpset file *after* the first time you ran the conversion. **Mif2Go** uses the first topic ID in the first file in the book. If all else fails, **Mif2Go** uses the base name of the FrameMaker book (or single FrameMaker file).

11.3.8.3 Deciding whether to rewrite the helpset file

Mif2Go creates a helpset file for you the first time you run a JavaHelp conversion. Although **Mif2Go** rewrites .jhm and .xml files every time you run the conversion, by default **Mif2Go** does not rewrite the helpset file during subsequent conversion runs.

To have **Mif2Go** rewrite the helpset file every time:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; WriteHelpSetFile = No (default) or Yes (write each time)
WriteHelpSetFile = Yes
```

Set WriteHelpSetFile=Yes if you move or delete the helpset file from the project directory every time you run the conversion.

Use the default value, WriteHelpSetFile=No, if you customize the helpset file outside of **Mif2Go**; see §11.3.8.5 [Adding custom helpset sections for JavaHelp 2](#) on page 383.

11.3.8.4 Providing a “favorites” option for JavaHelp 2

If you are using **Mif2Go** to generate JavaHelp 2, you can include in the helpset a provision for a “favorites” facility that allows the user to add topics to a “favorites” list:

```
[JavaHelpOptions]
; UseFavorites = No (default) or Yes (affects HelpSet File rewrite)
UseFavorites = Yes
```

11.3.8.5 Adding custom helpset sections for JavaHelp 2

JavaHelp 2 supports additional entries in the helpset file, such as an <impl> section; see the *JavaHelp System User's Guide* for information about this feature.

To add custom entries to the helpset, list the code for each entry in the following configuration-file section. For example:

```
[JH2_HelpsetAddition]
; Optional section used for literal additions to the JH2 <helpset>
<impl>
  <helpsetregistry helpbrokerclass="javax.help.DefaultHelpBroker" />
  <viewerregistry viewertype="text/html"
    viewerclass="com.sun.java.help.impl.CustomKit" />
  <viewerregistry viewertype="text/xml"
    viewerclass="com.sun.java.help.impl.CustomXMLKit" />
</impl>
```

You can put anything you please in section [JH2_HelpsetAddition], and whatever you add is included in the helpset file; for example, you can add your own <presentation> sections.

If **Mif2Go** does not rewrite the helpset file each time (see §11.3.8.3 [Deciding whether to rewrite the helpset file](#) on page 383), you could simply add custom sections directly to the helpset file.

11.3.8.6 Requiring full paths in the helpset file

To specify full paths instead of simple file names in links from the helpset file:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; HSPathNames = No (default, strip path from filenames)
; or Yes (use full path)
HSPathNames = Yes
```

The default setting, HSPathNames=No, is almost always the correct value, because the navigational files referenced in the helpset file are almost always in the same directory

(see §11.3.7.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 378). The only reason to put navigational files elsewhere would be to accommodate a proprietary standard; *this setting is intended to support that use only.*

11.3.9 Coping with JavaHelp / Oracle Help viewer limitations

JavaHelp viewer limitations and defects are described in the *JavaHelp System User's Guide*. **Mif2Go** provides workarounds for some; others you will have to put up with. The Oracle Help viewer has a different set of limitations. Some known limitations:

[Special characters in JavaHelp](#)

[Anchor tags in JavaHelp](#)

[Image size units in JavaHelp](#)

[CSS in JavaHelp or Oracle Help](#)

[Index entries.](#)

*Special
characters in
JavaHelp*

Some characters with ANSI decimal values in the range 128 through 159 do not display properly. For example, regular bullet characters (ANSI 149) show as small boxes in the JavaHelp viewer, unless you map them to a different value, with a setting such as the following:

```
[CharConvert]
149 = <b>&#183;</b>
```

See §13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454 and §21.5 [Assigning properties to text formats](#) on page 653.

*Anchor tags in
JavaHelp*

Each anchor tag in HTML, including the <a> tag produced from each marker in your FrameMaker document, is replaced by a space in the JavaHelp viewer. There is no feasible workaround for this defect. **Mif2Go** usually produces more than one , and <a> tags cannot be nested. Placing all <a> tags before the opening <p> eliminates the spaces, but adds a blank line above, which is even worse.

*Image size units
in JavaHelp*

A px suffix on image width and height attribute values causes the JavaHelp viewer to show the image as a thumbnail; so for JavaHelp, by default **Mif2Go** omits the suffix. Make sure you do not override this default; see §23.9.5 [Specifying px units for graphics sized in pixels](#) on page 722.

*CSS in JavaHelp
or Oracle Help*

Support for CSS is limited (in different ways) in the JavaHelp and Oracle Help viewers. You might have to resort to font tags and alignment attributes instead of using a style sheet. See §21.7.4 [Including or excluding font tags](#) on page 665.

JavaHelp CSS does not respect the list-style rule; therefore, by default, **Mif2Go** adds the type attribute to list wrappers ol and ul. To omit the type attribute from list wrappers:

```
[CSS]
; UseListTypeAttribute = Yes (default for JavaHelp, to fix CSS bug)
; or No (default for other formats, go by NoAttribLists value)
UseListTypeAttribute = No
```

See §21.12.2.7 [Including or excluding the type list attribute](#) on page 678.

Index entries

Index entries have limitations in both viewers; see §11.4.3 [Configuring index entries for JavaHelp or Oracle Help](#) on page 386.

11.3.10 Compiling JavaHelp with Helen

If you intend to compile your JavaHelp project with third-party compiler Helen, specify the following option:

```
[JavaHelpOptions]
; Helen = No (default) or Yes (to account for quirks in Helen for JH)
Helen = Yes
```

Helen might not like some valid HTML constructs.

11.4 Generating contents and index

To understand whether, how, and when **Mif2Go** generates contents and index files for JavaHelp and Oracle Help, see §7.3.4.1 [Choosing contents and index methods for HTML-based Help](#) on page 207.

In this section:

- §11.4.1 [Configuring contents entries for JavaHelp or Oracle Help](#) on page 385
- §11.4.2 [Assigning TOC images and expansion levels in JavaHelp 2](#) on page 385
- §11.4.3 [Configuring index entries for JavaHelp or Oracle Help](#) on page 386
- §11.4.4 [Eliminating index-marker artifacts from text](#) on page 386
- §11.4.5 [Locating JavaHelp or Oracle Help contents and index files](#) on page 387

See also:

- §7.3 [Producing contents and index for Help systems](#) on page 204
- §7.4 [Configuring contents entries for Help systems](#) on page 209
- §7.5 [Configuring index entries for Help systems](#) on page 211

11.4.1 Configuring contents entries for JavaHelp or Oracle Help

Headings that start topics, or to which you assign the `Contents` property or a contents level, are automatically included in the contents; see the following:

- §7.4.3 [Including contents entries in HTML-based Help](#) on page 209.
- §7.4.4 [Setting contents levels for HTML-based Help](#) on page 210.

However, if you set the following option, links might be missing for contents entries that are not topic headings:

```
[JavaHelpOptions] OR [OracleHelpOptions]
RemoveInternalAnchors = Yes
```

This is mainly an issue for JavaHelp, where anchors in text cause unwanted spacing; see §11.9 [Linking to destinations within topics](#) on page 399.

11.4.2 Assigning TOC images and expansion levels in JavaHelp 2

Mif2Go supports several JavaHelp 2 `<toc>` and `<tocitem>` attributes that allow you to control which TOC levels are expanded and collapsed, and what images are displayed in the TOC tree.

To set the `<tocitem>` expand attribute by level, assign `Yes` (expand) or `No` (collapse) to each level number:

```
[TocLevelExpand]
; The JH default is to expand only top-level (1) items; this sets the
; <tocitem> expand attribute according to level.
1 = Yes
2 = No
```

The JavaHelp default is to expand only top-level (level 1) items.

To designate images to be displayed in the TOC tree, assign an image ID to each different TOC item. For example:

```
[JavaHelpOptions]
; The JH default is to use the Toc*Image graphics for all levels; this
; replaces those graphics by setting the <tocitem> image attributes.
; The image IDs are mapped in [JHImages] as usual
TocClosedImage = closedsign
TocOpenImage = opensign
TocTopicImage = topicicon
```

The JavaHelp default is to use the assigned Toc*Image graphics for all TOC levels. However, to use the same graphic for all TOC items at a given level, you can assign an image ID to the level number. For example:

```
[TocLevelImage]
; The image IDs are mapped in [JHImages]
1=overview
```

You must map each image ID to the location of its corresponding graphic in section [JHImages]; see §11.8.1.2 [Mapping image names to graphics files](#) on page 394.

11.4.3 Configuring index entries for JavaHelp or Oracle Help

Mif2Go generates the JavaHelp or Oracle Help index file, *MyDocIndex.xml*. However, most of the ways you can customize index entries for HTML-based help, described in §7.5 [Configuring index entries for Help systems](#) on page 211, *do not work* for either JavaHelp or Oracle Help. Both produce indexes with the following possibly undesirable display features:

- Index entries are not formatted; any character formatting is lost.
- Sort-order settings from the **Mif2Go** configuration file are ignored, even though they correctly inform the order of items in *MyDocIndex.xml*.

JavaHelp In JavaHelp there is no graceful way to handle index entries that have multiple references to different places in the helpset, so **Mif2Go** converts multiple references into subentries, each with the topic title (including any autonumber) as the visible information.

Oracle Help for Java Oracle Help supports only two levels of index entries; the index view collapses any levels beyond the second.

Oracle Help provides an `<iindexentry>` tag for index files, which affects how topic names display for multiple references to the same index term. You can turn it off:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; UseIndexentryTag = Yes (OracleHelp only, default)
; or No (as in Sun JavaHelp)
UseIndexentryTag = No
```

You should not need to change the default setting for `UseIndexentryTag`, unless you are generating Oracle Help for Java *and* you prefer the way **Mif2Go** handles multiple-reference index entries for JavaHelp.

See §7.5 [Configuring index entries for Help systems](#) on page 211.

11.4.4 Eliminating index-marker artifacts from text

When you use a JavaHelp viewer, if you see question marks in the helpset where **Index** markers occur in your FrameMaker document, you can eliminate these artifacts with the following settings:

```
[Markers]
Index = NoIndex
```



```
[MarkerTypes]
NoIndex = Delete
```

See §29.3 [Remapping marker types and hypertext commands](#) on page 836.

Because **Mif2Go** produces the JavaHelp index file, you can still get an index based on those markers; see §7.5 [Configuring index entries for Help systems](#) on page 211.

11.4.5 Locating JavaHelp or Oracle Help contents and index files

When **Mif2Go** generates contents and index for JavaHelp or Oracle Help, you end up with the following files:

- a contents file, *MyProjTOC.xml*
- an index file, *MyProjIndex.xml*.

These files must reside in the same directory as the helpset file (*MyDoc.hs*), usually the `help` directory; see §11.3.7.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 378.

11.5 Providing full-text search for JavaHelp / Oracle Help

Including full-text search capability in the Help system for either JavaHelp or Oracle Help requires using an external indexing program to create a search index, and providing a link to the search index from the helpset file.

In this section:

§11.5.1 [Including a search-index link in the helpset file](#) on page 387

§11.5.2 [Creating a search index for JavaHelp](#) on page 388

§11.5.3 [Creating a search index for Oracle Help](#) on page 389

11.5.1 Including a search-index link in the helpset file

To indicate that you want to include full-text search (FTS) capability for JavaHelp or for Oracle Help:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; UseFTS = Yes (default) or No (affects HelpSet File rewrite)
UseFTS = Yes
```

When `UseFTS=Yes`, **Mif2Go** includes information in the helpset file to access a search index. You also have to run an indexing program to create the search index from the HTML output files. You can have **Mif2Go** run the program, or you can run it yourself; see:

§11.5.2 [Creating a search index for JavaHelp](#) on page 388

§11.5.3 [Creating a search index for Oracle Help](#) on page 389

Note: If `UseFTS=Yes` but the indexing program is *not* run, the helpset link to the search index could cause a run-time error.

When `UseFTS=No`, **Mif2Go** does not include a link to the search index. If you run the indexing program anyway, you get a search index, but the display might ignore it. However, if you use the Oracle Help for Java Helpset Authoring Wizard to produce a search index for Oracle Help, the Authoring Wizard itself provides a link in the helpset file.

11.5.2 Creating a search index for JavaHelp

JavaHelp utility program JHIndexer creates a search index for JavaHelp, and places the search index in a subdirectory called `JavaHelpSearch`. You can have **Mif2Go** run JHIndexer, or you can run it yourself. In addition to creating a search index, you must also provide a link to the search index from the helpset file; see §11.5.1 [Including a search-index link in the helpset file](#) on page 387.

*Let **Mif2Go**
create FTS*

To have **Mif2Go** automatically run JHIndexer:

```
[JavaHelpOptions]
; FTSCCommand = for Sun Java Help, path to jhindexer, such as:
FTSCCommand = D:/jh2.0_01/jh2.0/javahelp/bin/jhindexer
```

The value of `FTSCCommand` must include an absolute path to the directory where the JHIndexer program is installed on your system. If the path includes spaces, you must enclose it in double quotes. For example:

```
[JavaHelpOptions]
FTSCCommand = "G:/JH/jh2.0_01/jh2_0/javahelp/bin/jhindexer"
```

Do not enclose parameters in quotes:

- Use backslashes as separators in path-name parameters.
- Use a dash (“-”) instead of a forward slash to prefix a command option.

When you specify a value for `FTSCCommand`, after generating output files for JavaHelp, **Mif2Go** first removes any search-index directory previously created by JHIndexer, then uses the command to run JHIndexer and produce a new search index.

If you also specify a value for `[Automation]WrapPath`, **Mif2Go** copies all needed files from the project directory to the JavaHelp directory structure before running JHIndexer; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

You might also want to set the following option, so you can see any error messages that result:

```
[Automation]
; KeepCompileWindow = No (default)
; or Yes (so any error messages can be seen)
KeepCompileWindow=Yes
```

When `KeepCompileWindow=Yes`, a system window opens when the indexer runs. If there are no indexer errors, you will see only a command prompt when indexing finishes. You must dismiss the window before **Mif2Go** can continue processing.

*Create FTS
yourself*

If you do not specify a value for `FTSCCommand`, you must run JHIndexer yourself, from the directory where your helpset file is located, and specify the directory where the HTML files are located. For example, at a Windows command prompt:

```
D:
cd \path\to\MyOutput\help
del /f /q JavaHelpSearch
path\to\JavaHelp\files\bin\jhindexer html
```

If you are using a directory structure such as the following for your project, run JHIndexer from the `Help` directory:

```
MyDoc      (FrameMaker files)
..MyOutput (Mif2Go files)
...Help    (JHM, HS, TOC, Index; run JHIndexer from this directory)
.....HTML
.....Graphics
```

Your structure ends up looking like this:

```

MyDoc      (FrameMaker files)
..MyOutput (Mif2Go files)
...Help    (JHM, HS, TOC, Index)
.....JavaHelpSearch (Created by JHIndexer)
.....HTML
.....Graphics

```

JavaHelp search caveats

It is best to remove any previous search-index directory before you run JHIndexer to create a new directory. If a previous attempt to create a search index failed, further attempts will also fail if the failed search-index directory is present.

You must run JHIndexer from the directory where the helpset file is located. If you try to run JHIndexer from within the HTML directory, JHIndexer will put the JavaHelpSearch directory *inside* the HTML directory. If you try to run it from any other directory, you get strange effects in the Contents panel.

See also:

§7.2.4 [Compiling and distributing Help systems](#) on page 204

§11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379

§11.3.7.4 [Specifying a path for search-index links](#) on page 381

§11.5.1 [Including a search-index link in the helpset file](#) on page 387

§35.10 [Gathering and processing Help-system files](#) on page 971

11.5.3 Creating a search index for Oracle Help

Oracle Help for Java utility program Indexer creates a search index for Oracle Help. Indexer generates an index called `myproj.idx`, placed in the same directory as the helpset file, `myproj.hs`. Before running Indexer, make sure the CLASSPATH environment variable on your system includes a path to Indexer; see §11.3.6 [Establishing an Oracle Help environment](#) on page 377.

You can create a search index for Oracle Help in any of the following ways:

§11.5.3.1 [Directing Mif2Go to create full-text search for Oracle Help](#) on page 389

§11.5.3.2 [Creating full-text search for Oracle Help via command line](#) on page 390

§11.5.3.3 [Using the Oracle Help Wizard to create full-text search](#) on page 390

See also:

§7.2.4 [Compiling and distributing Help systems](#) on page 204

§11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379

§11.5.1 [Including a search-index link in the helpset file](#) on page 387

§35.10 [Gathering and processing Help-system files](#) on page 971

11.5.3.1 Directing Mif2Go to create full-text search for Oracle Help

To have **Mif2Go** run Indexer for you:

```

[OracleHelpOptions]
; FTSCCommand = for Oracle Help, indexing command, typically:
FTSCCommand = java -mx256m oracle.help.tools.index.Indexer

```

When you specify a value for FTSCCommand, **Mif2Go** uses the command you supply to run Indexer after generating output files for Oracle Help. If your project is very large, you might want to increase the value of the `-mx` option. If the Indexer command includes spaces, you must enclose it (but not the parameters) in double quotes. Prefix options with a dash (“-”) rather than a forward slash.

You might also want to set the following option, so you can see any error messages that result:

```
[Automation]
; KeepCompileWindow = No (default)
; or Yes (so any error messages can be seen)
KeepCompileWindow=Yes
```

When `KeepCompileWindow=Yes`, a system window opens when the indexer runs. If there are no indexer errors, you will see only a command prompt when indexing finishes. You must dismiss the window before **Mif2Go** can continue processing.

If you specify a value for `[Automation]WrapPath`, **Mif2Go** copies all needed files from the project directory to the Oracle Help directory structure before running Indexer; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

11.5.3.2 Creating full-text search for Oracle Help via command line

You can run Indexer yourself, from the directory where your helpset file is located. To run Indexer directly from a `.bat` file, include the following commands:

```
cd /D path\to\hs
REM The following command must be typed all on one line:
java -mx256m oracle.help.tools.index.Indexer path\to\hs myproj.idx
```

Paths should be absolute rather than relative. If a path includes spaces, you must enclose it (but not the parameters) in double quotes (“ ”) rather than a forward slash.

11.5.3.3 Using the Oracle Help Wizard to create full-text search

You can use the Oracle Help for Java HelpSet Authoring Wizard to generate a full-text search index. According to the Oracle Help for Java User Guide:

When you install OHJ on Windows, a batch file and an initialization file for starting the wizard are generated, using the path into which you installed OHJ. A shortcut for starting the wizard is also installed on the Windows Start menu. Select this shortcut to start the wizard. Alternatively, you can issue the following command at the command prompt:

```
OHJ_path\launcher.exe OHJ_path\bin\authoringWizard.ini
```

Follow the prompts in the wizard.

Start the Wizard in the same directory as the helpset file. After browsing for the helpset file in step 1, accept the defaults the Wizard presents for steps 2 through 9; it is best not to stray from the **Next** path. Step 10 requires a value for **Base Name**: enter the base name of your helpset file, make sure you allow the Wizard to create a backup, and click **Finish**.

Next, delete `myproj.hs`, and rename `myproj.hs.BAK` to `myproj.hs`. The result should be a fully functional search index.

11.6 Creating and viewing a Java Archive (JAR) file

To deploy a JavaHelp system, it is best to archive all the required components in a single executable JAR file. Although you can create a JAR file for an Oracle Help system, the result probably will not be executable, at least on Windows.

In this section:

§11.6.1 [Creating a JAR file](#) on page 391

§11.6.2 Viewing a JAR file on page 391

11.6.1 Creating a JAR file

To create a JAR file you need archiving program `jar.exe`, which is included in the Java Software Development Kit. You can download the JDK here:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

You have two choices:

[Let Mif2Go create the JAR file](#)

[Create the JAR file yourself.](#)

*Let **Mif2Go**
create the JAR
file*

To have **Mif2Go** create a JAR file for you, specify a **jar** command that works on your system. For example:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; JarCommand = path to jar, without any parameters; the cvf and * are
; added before and after the HSFileName (with .jar ext) automatically
JarCommand = D:/j2sdk14/jdk/bin/jar
```

JarCommand must include an absolute path to `jar.exe`, unless `jar.exe` is on your system PATH. Do not include parameters. **Mif2Go** provides the `cvf` and `*` parameters, and gives the resulting JAR file the base name of your helpset file as specified by `HSFileName` (see §11.3.8.1 [Specifying helpset file name and title](#) on page 382), and extension `.jar`.

*Create the JAR
file yourself*

To create a JAR file yourself, run the **jar** command from the directory where your helpset file is located. The following example creates file `myproj.jar` and places it in the same directory as the helpset file. You can put the commands in a `.bat` file.

```
cd \path\to\MyOutput\help
path\to\JDK\files\bin\jar cvf myproj.jar *
```

11.6.2 Viewing a JAR file

To view a JavaHelp JAR file (for example, `myproj.jar`):

1. Open a Command Prompt window.
2. Navigate to the directory where `java.exe` is located (see §11.3.5 [Establishing a JavaHelp environment](#) on page 377).
3. Execute the following command (which must be all on one line):

```
java -jar path\to\hsvviewer.jar -helpset path\to\myproj.jar
```

You can include navigation and execution commands in a `.bat` file for convenience. For example:

```
cd C:\Program Files\Java\j2re1.4.2_03\bin
java -jar G:\jh20\demos\bin\hsvviewer.jar -helpset g:\jh\myproj.jar
```

*JAR files might
have broken links*

If your document includes cross references or hypertext links that contain file names that do not match the case of the target file name, the links will not work when you view the JAR file. You can correct this problem either of the following ways:

- Change all file names in your FrameMaker document to lowercase, and set `[HTMLOptions]MakeFileHrefsLower=Yes`; see §19.2.6 [Forcing link text to lowercase](#) on page 613.
- Inspect every cross-reference and hypertext marker in your document, and either fix the markers or change the case of the FrameMaker file name.

On Windows you can get around the case mismatch problem by viewing the `.hs` file instead of the `.jar` file.

*Viewing an
Oracle Help JAR
file is problematic*

Although creating a JAR file for an Oracle Help project appears to work correctly, at Omni Systems we have not succeeded in viewing the resulting JAR file on Windows. On the other hand, Oracle Help .hs files can be viewed successfully on Windows.

See also:

§11.3.6 [Establishing an Oracle Help environment](#) on page 377.

11.7 Converting a glossary to JavaHelp 2

If your FrameMaker document includes a glossary that consists of alternating terms and their definitions, you can take advantage of JavaHelp 2 glossary support.

In this section:

§11.7.1 [Evaluating glossary usability](#) on page 392

§11.7.2 [Assigning glossary properties](#) on page 392

§11.7.3 [Configuring glossary IDs](#) on page 392

§11.7.4 [Eliminating glossary entries from the JavaHelp TOC](#) on page 393

11.7.1 Evaluating glossary usability

The JavaHelp 2 glossary system is designed to work with a separate file for each glossary term. The terms are listed in an index-style glossary navigation pane; clicking a term in the glossary pane opens a small window that displays the definition. There are no links to definitions from terms that appear in topics.

If you provide a single Glossary topic that contains all the terms and definitions, instead of using the built-in JavaHelp 2 glossary, you can include cross references to the terms wherever needed. You cannot insert jumps from topics to the JavaHelp 2 glossary.

11.7.2 Assigning glossary properties

To convert a FrameMaker glossary to a JavaHelp 2 glossary:

```
[JavaHelpOptions]
; UseGlossary = No (default) or Yes (affects HelpSet File rewrite)
UseGlossary = Yes
```

Use two paragraph formats for glossary entries in FrameMaker: one format for the term, and one for the definition. For example, *Gterm* for the term, and *Gdef* for the definition.

Assign property GlossTerm to each glossary-term paragraph format. For example:

```
[HTMLParaStyles]
; doc style (para or char) = keywords for functions and properties
; GlossTerm is used for JavaHelp 2 only, to identify para formats
; used for glossary terms (where they are defined in the next
; following paragraph).
Gterm = GlossTerm
```

Because every glossary term is followed by a definition, you do not have to assign a property to the definition format.

11.7.3 Configuring glossary IDs

For each glossary term, **Mif2Go** creates an ID that consists of a special prefix, the term itself (omitting or replacing any spaces), and an optional suffix.

To specify glossary-term prefix, suffix, and space replacer:


```
[JavaHelpOptions]
; GlossPrefix = prefix used to form glossary term IDs, default GLO_
GlossPrefix=GLO_
; GlossSuffix = suffix used to form glossary term IDs, default none
GlossSuffix=
; GlossSpace = replacement for spaces in glossary term IDs,
; default none
GlossSpace =
```

Mif2Go stores glossary terms, IDs, and glossary file names (allowing for splitting a glossary file) in a *ChapName.bhg* file for each FrameMaker file in your document.

Mif2Go uses the set of *.bhg files to do the following:

- generate glossary.xml
- add glossary IDs and *filename#GLO_term* entries to the .jhm file
- add a glossary <view> to the helpset file.

11.7.4 Eliminating glossary entries from the JavaHelp TOC

Material that **Mif2Go** converts to a JavaHelp 2 glossary no longer appears in the resulting JavaHelp 2 system as a regular topic. Therefore you might find an orphaned glossary entry in the JavaHelp TOC, pointing to a topic that contains only the original heading of your FrameMaker glossary.

To eliminate a TOC entry for the glossary, do either of the following:

- Use conditional text and a conversion template to hide the entry in FrameMaker; see §2.4 [Importing formats from a conversion template](#) on page 67.
- Create a distinct paragraph format for the heading in FrameMaker, and in the configuration file assign [HTMLParaStyles] property Delete to that format; see §21.3.12 [Eliminating unwanted paragraphs](#) on page 652.

If you use the second method, you must also remove any cross references to the heading from other topics.

11.8 Defining windows for JavaHelp or Oracle Help

In this section:

- §11.8.1 [Specifying window parameters for JavaHelp 2](#) on page 393
- §11.8.2 [Specifying window parameters for Oracle Help](#) on page 398
- §11.8.3 [Jumping to secondary windows in JavaHelp or Oracle Help](#) on page 399

11.8.1 Specifying window parameters for JavaHelp 2

Mif2Go supports window definitions for JavaHelp 2. For JavaHelp 1, you have to roll your own; see the *JavaHelp System User's Guide*. For Oracle Help for Java, see §11.8.2 [Specifying window parameters for Oracle Help](#) on page 398.

In this section:

- §11.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 394
- §11.8.1.2 [Mapping image names to graphics files](#) on page 394
- §11.8.1.3 [Understanding JavaHelp 2 window-access limitations](#) on page 395
- §11.8.1.4 [Specifying window-access object properties](#) on page 395
- §11.8.1.5 [Overriding window-access properties with markers](#) on page 397
- §11.8.1.6 [Designing your own window-access marker names](#) on page 398

11.8.1.1 Assigning default window parameters for JavaHelp 2

Assign a name to each JavaHelp 2 window type you expect to define. For example:

```
[JavaHelpOptions]
; Windows = list of JH2 windows, each defined by its own section
Windows = mainwin screenshot procwin
```

By default, the first name listed is the name of the main window.

Note: Window name `popup` is a reserved name that identifies the window as a pop-up window; see §7.8 [Creating pop-up topics for Help systems](#) on page 225. Do not list `popup` as a window type.

For each window name assigned to `[JavaHelpOptions]Windows`, specify parameters for that window type in a separate configuration-file section of the same name as the window. These parameters inform the window descriptions **Mif2Go** places in the JavaHelp 2 helpset file. For example:

```
[JavaHelp window name] (such as [mainwin] or [secwin])
; Default = No (default) or Yes (to make this the default window)
Default = Yes
; Name = name used to reference in code
Name = mainwin
; Title = name in title bar
Title = Mif2Go User's Guide
; Top = top edge, pixels from top of screen, default 200
Top = 200
; Left = left edge, pixels from left side of screen, default 200
Left = 200
; Height = height in pixels, default 400
Height = 400
; Width = width in pixels, default 400
Width = 400
; NavPane = Yes (default, with toolbar is tripane)
; or No (for secondary windows)
NavPane = Yes
; NavIcons = Yes (default) or No (show text instead)
NavIcons = Yes
; Image = image ID, mapped in [JHImages] if used
Image = mainwinimage
; Toolbar = list of items to include, from: Back, Forward, Home,
; Reload, Favorites (add current page to), Print, PrintSetup,
; Separator (on bar).
Toolbar = Home Back Forward Separator Print Separator Favorites
; Optional images for toolbar items, itemImage=image file ID,
; mapped in the [JHImages] section
HomeImage = house
```

Note: Size and position settings for secondary windows (Top, Left, Height, and Width) are always overridden by object properties of the links to those windows; see §11.8.1.4 [Specifying window-access object properties](#) on page 395.

11.8.1.2 Mapping image names to graphics files

If you assign names to image parameters for specific windows (such as window `Image` or toolbar `HomeImage`), map each image name to the location of its graphic file, relative to the location of the helpset file. For example:

```
[JHImages]
; image ID = path, relative to .hs file
mainwinimage = graphics/floral.gif
house = graphics/littlehouse.gif
```

11.8.1.3 Understanding JavaHelp 2 window-access limitations

In JavaHelp 2, pop-up links and jumps to secondary windows are represented as objects, placed at the start of their hotspots, rather than as conventional links. Only the objects themselves are active links. Hotspot text that you delimit with a character format in FrameMaker (see §7.8.2 [Defining a pop-up hotspot](#) on page 226) looks like a hotspot in JavaHelp 2, but has no effect.

The only way to include link-specific hotspot text in FrameMaker that both looks and acts like a hotspot in JavaHelp 2 is to insert in your document special markers that contain the hotspot text, plus (if necessary) additional special markers that designate font properties for hotspot text; see §11.8.1.5 [Overriding window-access properties with markers](#) on page 397.

In other words, for an active-link text hotspot, you have to use markers to recreate any text for the hotspot that might already be present in the document. If you can, that is; underlines, for example, are not possible. For this reason, the *default* window-access object **Mif2Go** produces is not a text object, but instead a button that immediately precedes text that is already designated as a hotspot.

11.8.1.4 Specifying window-access object properties

You specify properties for window-access objects by assigning values to object-property keywords in section [JavaHelpOptions]. [Table 11-2](#) on page 396 and [Table 11-3](#) on page 396 list the keywords, the values you can assign to each keyword, and the default when you do not assign a value.

In this section:

§11.8.1.4.1 [Changing window type, size, or position via access object](#) on page 395

§11.8.1.4.2 [Specifying link properties via window-access object](#) on page 396

11.8.1.4.1 Changing window type, size, or position via access object

For window type, size, and position you can do the following:

[Specify pop-up window size but not position](#)

[Specify secondary window size and position](#)

[Override secondary window type](#)

*Specify pop-up
window size but
not position*

Only one type of pop-up window can be defined, so in the absence of overrides, all pop-up window-access properties apply to all pop-up links and windows. The only way to specify different sizes for different pop-up windows is by inserting special markers before individual pop-up hotspots; see §11.8.1.5 [Overriding window-access properties with markers](#) on page 397.

Pop-up window position is not configurable; a pop-up window always pops up immediately under the link.

*Specify
secondary
window size and
position*

The secondary-window size and position settings listed in [Table 11-2](#) override the default size and position parameters in the helpset file for *all* secondary windows, making the default values moot. See §11.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 394. The only way to configure different sizes and positions for different secondary window types is by inserting special markers before each jump; see §11.8.1.5 [Overriding window-access properties with markers](#) on page 397.

*Override
secondary
window type*

The default value for [JavaHelpOptions]SecName is empty (see [Table 11-2](#)). If you do not include a setting for SecName, for each secondary-window jump the properties specified for that jump in [SecWindows] apply. If you specify a value for SecName, then

for all secondary-window jumps, any properties (except size and position) you assign to the SecName window type in its own configuration section override the corresponding properties of whatever window type is specified for the jump in [SecWindows]. See:

§7.7 [Jumping to secondary windows in Help systems](#) on page 224

§11.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 394

Table 11-2 [JavaHelpOptions] pop-up and secondary window properties

Window	Keyword	Value	Default	Comments
Pop-up	PopSize	<i>width,height</i>	250, 300	One comma, no space between pixel values; these settings override corresponding helpset parameters
Secondary	SecSize	<i>width,height</i>	250, 300	
	SecLocation	<i>left,top</i>	600, 200	
	SecName	<i>window name</i>	Default is the secondary window assigned in [SecWindows] to a given jump-hotspot format	

11.8.1.4.2 Specifying link properties via window-access object

You can specify a button, a text string, or a graphic for JavaHelp 2 to display as a window-access object for a pop-up link or a secondary-window jump. If you specify a text string, you can assign values to font keywords to apply a limited amount of formatting. [Table 11-3](#) on page 396 lists the base keywords, and the values you can assign to those keywords, to configure the appearance of a window-access object.

Prefix base keywords with "Pop" or "Sec"

You must supply a prefix for each of the object-property base keywords listed in [Table 11-3](#), to indicate whether the keyword represents a property for a pop-up window link or for a secondary-window jump:

- For a pop-up window, prefix the keyword with Pop; for example:

```
[JavaHelp window name]
PopType = Graphic
PopGraphic = ../graphics/popicon.gif
```

- For a secondary window, prefix the keyword with Sec; for example:

```
[JavaHelp window name]
SecType = Button
```

Table 11-3 [JavaHelp window name] window-access object properties

Keyword*	Value	Default value and comments
Type	Button Graphic Text	When Type=Button (default), the value of keyword Text is the label When Type=Graphic, the value of keyword Text (or keyword Graphic) is the location of the image file When Type=Text, Font* properties apply to the value of keyword Text
Graphic	<i>URL for image file</i>	Default is ../graphics/lp.gif When Type =Graphic, value is the relative URL of the GIF or JPEG file
Text	<i>plain text</i> < > &	Default is > When Type=Text, Font* properties apply to the value of keyword Text When Type=Graphic, value can be the location of the image (which can be specified as a value either for keyword Text or for keyword Graphic)
FontFamily	SansSerif Serif Monospaced Symbol Dialog DialogInput	Pop-up window default is SansSerif Secondary window default is Serif Not all FontFamily values work the way you might expect them to work

Table 11-3 [*JavaHelp window name*] window-access object properties

Keyword*	Value	Default value and comments
FontSize	xx-small, x-small, small (default), medium, large, x-large <i>index number</i> bigger smaller + <i>n</i> - <i>n</i> <i>nnpt</i>	<i>index number</i> is a plain digit bigger increases the size by one index smaller decreases the size by one index + <i>n</i> increases the size by <i>n</i> - <i>n</i> decreases the size by <i>n</i> <i>nnpt</i> specifies the font size in points
FontWeight	plain (default), bold	
FontStyle	plain (default), italic	
FontColor	blue (default), black, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow	
* Prefix the base keyword with one of the following: Pop for a pop-up window property Sec for a secondary-window property		

*Eschew
Type=Text*

Despite the limited font-tweaking possibilities listed in [Table 11-3](#), you might want to avoid setting PopType=Text or SecType=Text, unless you are happy with a link that consists of a single >, <, or & character, or a little box (what you get when, for example, you specify PopFontFamily=Symbol and choose a character from the Symbol font). See §11.8.1.3 [Understanding JavaHelp 2 window-access limitations](#) on page 395.

*Multiple markers
for each hotspot*

To actually create a text hotspot with context-specific content, you would have to insert a collection of markers, all different, before every pop-up link or secondary-window jump in a file, to handle the varying text content and properties—to the limited extent that you can do so. See §11.8.1.5 [Overriding window-access properties with markers](#) on page 397.

*Not all “special”
characters work*

Keep in mind that JavaHelp does not support the “undefined” characters with ASCII decimal values from 128 through 159, even though these characters are used heavily in Windows for quotes, bullets, and so forth. See §13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454.

11.8.1.5 Overriding window-access properties with markers

To change a JavaHelp 2 object property for a specific link that accesses a pop-up or secondary window, you can insert a special marker somewhere in your document before the link you want to tweak.

Marker name

The name of the marker is the keyword for the property to be tweaked, prefixed by **JH2Pop** for a pop-up window or **JH2Sec** for a secondary window, or by a prefix you specify; see §11.8.1.6 [Designing your own window-access marker names](#) on page 398. The keyword part of the marker name can be any of the following:

- **Size** (see [Table 11-2](#)); for example, **JH2PopSize** or **JH2SecSize**
- **Name** or **Location**, for secondary windows only (see [Table 11-2](#)): **JH2SecName**, **JH2SecLocation**
- the base name of a window-object access keyword listed in [Table 11-3](#); for example, **JH2PopText** or **JH2SecGraphic**.

Marker content

The content of the marker is any value you could assign to the keyword in section [JavaHelpOptions]; see §11.8.1.4 [Specifying window-access object properties](#) on page 395. The marker content overrides the corresponding keyword setting, but only for the next pop-up or secondary window link in the file.

Resize individual pop-up windows Use **JH2PopSize** markers to resize pop-up windows in JavaHelp 2 (something Oracle Help for Java does for you). For example, to specify the dimensions of one particular pop-up window, you could place a marker of type **JH2PopSize** with content 300,75 somewhere before the pop-up hotspot, for a 300-pixel-wide and 75-pixel-high pop-up window.

11.8.1.6 Designing your own window-access marker names

To specify JavaHelp 2 window-access marker-name prefixes other than **JH2Pop** and **JH2Sec** (see §11.8.1.5 [Overriding window-access properties with markers](#) on page 397):

```
[JavaHelpOptions]
; PopMarkerPrefix = prefix for pop-up window marker type,
; default JH2Pop
PopMarkerPrefix = JH2Pop
; SecMarkerPrefix = prefix for secondary window marker type
; default JH2Sec
SecMarkerPrefix = JH2Sec
```

Prefix required The window-access marker names require *some* prefix; if you try to assign an empty prefix, **Mif2Go** ignores the setting and uses the default value for that prefix.

Same keyword, different prefix To provide different window-access settings for the same files in different **Mif2Go** projects, you can insert two or more markers (with different contents) whose names have the same keyword suffix but different prefixes, then just specify the appropriate prefix in the configuration file to select a set of markers to use for a given project.

Correct a document-wide typo Another possible reason for designating your own prefixes: if you make a systematic error with the marker type name, such as using **JHPop*** instead of **JH2Pop***, you can avoid correcting the name in who knows how many FrameMaker files, a change that cannot be accomplished with a template.

11.8.2 Specifying window parameters for Oracle Help

Mif2Go puts Oracle Help for Java window descriptions into the .hs file when you provide parameters in the following section:

```
[OracleHelpWindows]
; Windowname = height,width,xpos,ypos,textcolor,linkcolor,background,
; buttons,title
; The first window listed becomes the default window.
Main = 50%,240,100,100,000000,0000ff,ffffff,c000,Main Help Window
```

List window properties in the order indicated, separated by commas. [Table 11-4](#) describes the properties you can specify for each window.

Table 11-4 Oracle Help for Java window properties

Property	Description
height	Height of window, in pixels or percent (indicated by suffix %)
width	Width of window in pixels or percent (indicated by suffix %)
xpos	Horizontal screen coordinate of upper left corner, in pixels
ypos	Vertical screen coordinate of upper left corner, in pixels
textcolor	RGB color of text in window, in hexadecimal
linkcolor	RGB color of links in window, in hexadecimal
background	RGB color of window background, in hexadecimal

Table 11-4 Oracle Help for Java window properties (continued)

Property	Description
buttons	Sum of the following hexadecimal values, in hexadecimal: 4 - Remove default buttons 40 - Add URL display 400 - Add Navigator button 2000 - Print 4000 - Back and Forward 8000 - Search 10000 - Dock and Undock
title	Display window title

*Do not define
pop-up windows
here*

Window name popup is a reserved name that identifies the window as a pop-up window; see §7.8 [Creating pop-up topics for Help systems](#) on page 225. Do not include an entry in [OracleHelpWindows] for a pop-up window, unless you really do not like the Oracle Help default yellow “sticky note” that pops up over the center of the parent window. A secondary window defined in [OracleHelpWindows] *replaces* the parent window.

11.8.3 Jumping to secondary windows in JavaHelp or Oracle Help

Use a character or paragraph format to define a hotspot for a jump to a secondary window, and assign the window name to that format:

```
[SecWindows]
; doc format = name of secondary window to use for jumps from
; within the span marked by this format (same as WinHelp usage)
ProcWindow = procwin
```

See §7.7 [Jumping to secondary windows in Help systems](#) on page 224. The window name popup is reserved for pop-up windows.

11.9 Linking to destinations within topics

After you convert a FrameMaker document, you might find that anchors are missing for links to destinations located within topic files. By default, for JavaHelp these anchors are suppressed, as a workaround for a JavaHelp 1 defect: very bad things happen in the JavaHelp viewer if you try to use destination anchors *within* topic files. In JavaHelp 1, you can safely use references only to the start of a topic file.

Internal references are not a problem in JavaHelp 2, nor in Oracle Help for Java. On the other hand, in JavaHelp, every anchor causes an extra space in text when you view output with the JavaHelp viewer.

To allow internal references for JavaHelp 1:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; RemoveInternalAnchors = Yes (JavaHelp 1 default, avoid JavaHelp bug)
; or No (Oracle Help and JavaHelp 2 default)
RemoveInternalAnchors = No
```

11.10 Creating ALinks for Oracle Help

An ALink keyword in Oracle Help for Java must consist of a single term; no spaces or punctuation. An ALink jump can specify only one ALink keyword.

Oracle Help for Java uses a link file for ALinks (regular JavaHelp does not support ALinks). **Mif2Go** creates the ALink file for Oracle Help, by default. However, if you

know you are *not* using ALinks, you can save a little time and disk space by directing **Mif2Go** not to include this file:

```
[OracleHelpOptions]
; MakeALinkFile = Yes (default, include OracleHelp ALinks) or No
MakeALinkFile = No
```

You can determine whether ALinks go to the beginning of the referenced topic file, or to the beginning of the paragraph that contains the ALink keyword. The default is the beginning of the topic file:

```
[OracleHelpOptions]
; ALinkRefs = File (default) or Para (start of containing para)
ALinkRefs = File
```

For ways to include ALink keywords and ALink jumps in your FrameMaker document, see §7.6 [Providing related-topic links for Help systems](#) on page 219.

To create a “pool” of ALinks, where every instance serves both as a jump and a target, you can use a FrameMaker format combined with **Mif2Go** macros. For example, suppose you designate paragraph format *ALinkUse* for this purpose. You could create an ALink reference and get both an ALink keyword (which makes the current topic a member of the group) and a hotspot that calls up that list of topics:

```
[HTMLParaStyles]
; ALink uses the contents of the para for the value of the ALink
; Name parameter of an ALink object.
ALinkUse = ALink CodeBefore CodeAfter

[ParaStyleCodeBefore]
ALinkUse = <a href="alink:

[ParaStyleCodeAfter]
ALinkUse = ">Related Topics</a>
```

When you assemble ALink jumps using macros, you do not access **Mif2Go** code that automatically interprets the `alink` protocol (see §7.6.5.1 [Configuring ALink jumps](#) on page 223); what you build is passed through to Oracle Help unaltered.

11.11 Merging JavaHelp or Oracle Help systems

JavaHelp and Oracle Help for Java support limited merging of helpsets. You list subprojects in the main project helpset; the subproject information is appended to the main project information. In the contents, subprojects are listed one after the other, after the main project, and each subproject has to begin at the top level. Index entries for each subproject are appended to those for the main project, for JavaHelp; index merging is implemented somewhat better for Oracle Help.

To specify that helpsets are to be merged:

```
[JavaHelpOptions] OR [OracleHelpOptions]
; UseSubHelpSets = No (default) or Yes (requires [HelpMergePaths])
UseSubHelpSets = Yes
```

You must specify the path to the run-time location of each subproject helpset, relative to the main project helpset. For example:

```
[HelpMergePaths]
; subproject name = path to its files during use (not construction)
mysub = ../mysub/
```

Only the path is used, not the helpset name.

For more information, see the following:

Merging HelpSets in *JavaHelp System User's Guide*

Oracle Help Overview > Merged Helpsets in OHJ in *Oracle Help Guide*

See also:

§7.11 [Setting up a dynamic modular Help system](#) on page 241

11.12 Setting up CSH for JavaHelp or Oracle Help

For context-sensitive help, you insert symbolic IDs into your FrameMaker files as hypertext **newlink** markers (see §34.1.2 [Using markers to add links and instructions](#) on page 935), at the appropriate topic start points. **Mif2Go** puts these IDs in the .jhm map file for you.

By default, **Mif2Go** removes punctuation and spaces from **newlink** marker content. If you require symbolic IDs for CSH that contain characters such as periods, set the following option:

```
[HTMLOptions]
; UseRawNewlinks = No (default, remove punctuation, spaces)
; or Yes (as is)
UseRawNewlinks = Yes
```

*CSH map file:
needed?*

The way an application calls JavaHelp or Oracle Help determines whether you need a CSH map file; this is up to the application developers. You have to ask the developers how the application calls the Help system:

- If the developers use numbers, you need a CSH map file, and the developers will supply it. The map file lists a symbolic ID for each numeric ID.
- If the developers use names, you do not need a CSH map file; however, the developers must tell you what symbolic IDs they are using, or you must tell them what symbolic IDs to use.

*Non-CSH internal
map file*

A CSH map file comes from a developer, and relates numeric IDs that are used *in the application* to symbolic IDs. But JavaHelp and Oracle Help each have an internal map file with extension .jhm, which relates symbolic IDs used *in the Help system* to locations in the Help files, with *different* numeric IDs. These two map files and sets of numbers have nothing to do with each other.

See §7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239.

12 Generating Eclipse Help

Mif2Go produces the XML and HTML files needed to support the core functionality of Eclipse Help. This section addresses issues that are specific to generating Eclipse Help. HTML settings described in section 13 and sections 18 through 34 apply also. Topics include:

- §12.1 [Understanding how Eclipse Help works](#) on page 403
- §12.2 [Setting up an Eclipse Help project](#) on page 403
- §12.3 [Configuring Eclipse Help manifest files](#) on page 407
- §12.4 [Configuring contents and index for Eclipse Help](#) on page 411
- §12.5 [Configuring search properties for Eclipse Help](#) on page 415
- §12.6 [Merging Eclipse Help projects](#) on page 415
- §12.7 [Setting up CSH for Eclipse Help](#) on page 417
- §12.8 [Packaging Eclipse Help files](#) on page 419

See also:

- §7 [Producing on-line Help](#) on page 199

12.1 Understanding how Eclipse Help works

Eclipse Help for the Eclipse Platform is based on an XML table of contents that specifies the structure of the Help system and references content in standard XHTML files. An Eclipse Help plug-in minimally consists of a plug-in manifest file, `plugin.xml`, and a TOC (table of contents) file, `toc.xml`. The manifest provides information about the plug-in, such as name, ID, and version number. The TOC file is registered with the Eclipse Platform, using the `org.eclipse.help.toc` extension point.

Mif2Go supports the core functionality of Eclipse Help, including the following:

- generation of primary and secondary TOCs
- indexing (for later versions of Eclipse)
- infopops: the Eclipse version of CSH (Context-Sensitive Help).

12.2 Setting up an Eclipse Help project

In this section:

- §12.2.1 [Creating an Eclipse Help project](#) on page 403
- §12.2.2 [Choosing set-up options for an Eclipse Help project](#) on page 404
- §12.2.3 [Deciding where to locate configuration settings](#) on page 405
- §12.2.3 [Deciding where to locate configuration settings](#) on page 405
- §12.2.5 [Making sure links work in Eclipse Help](#) on page 406
- §12.2.6 [Disabling breadcrumb trails in Eclipse Help](#) on page 406

12.2.1 Creating an Eclipse Help project

To create an Eclipse Help project:

1. Create a project directory for HTML files, separate from the directory where your FrameMaker document is located.

2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
3. Name your Eclipse Help project, and browse to the project directory you created in [Step 1](#).
4. Choose output type Eclipse Help and click **OK**.
5. Check options in the *Set Up Eclipse Help Project* dialog (see §12.2.2 [Choosing set-up options for an Eclipse Help project](#) on page 404).
6. Use a text editor to edit the resulting `_m2eclipse.ini` configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).
7. **Important:** To make the configuration fit your usage of headings to start topics, pay special attention to sections [HelpContentsLevels] (see §7.2.1 [Checking automatic Help topic assignments](#) on page 203) and [HTMLParaStyles] (see §18.2 [Splitting files](#) on page 586).

12.2.2 Choosing set-up options for an Eclipse Help project

When you select Eclipse Help as the output type for a new project, the *Set Up* dialog shown in [Figure 12-1](#) opens. [Table 12-1](#) shows the corresponding settings in the configuration file. *You must edit configuration files to specify additional options.*

See also:

§3.4 [Choosing project set-up options](#) on page 79

§13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425

§7 [Producing on-line Help](#) on page 199

Figure 12-1 Set Up Eclipse Help Project

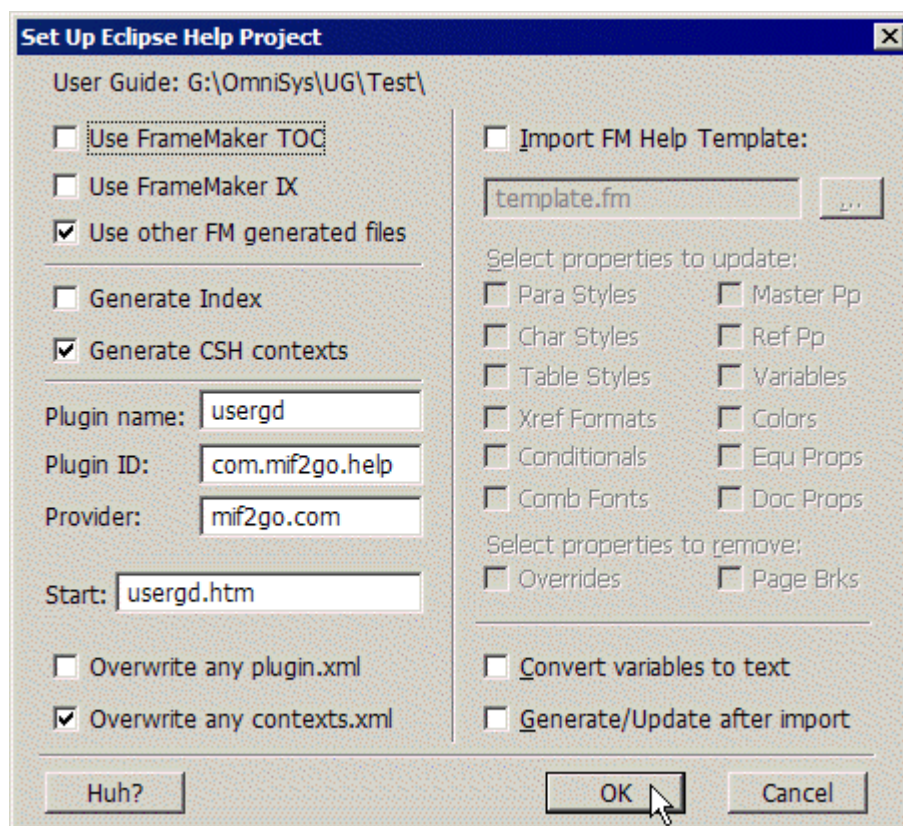


Table 12-1 Eclipse Help set-up options and configuration settings

Set-up dialog Option	Configuration file [EclipseHelpOptions] section Setting	Default	Ref.
Generate Index	UseIndex=Yes	No	12.4
Generate CSH contexts	UseContext=Yes	Yes	12.7
Plug-in name:	PluginName=Name	MyDoc	12.3.1
Plug-in ID:	PluginID=ID	com.mif2go.help	12.3.1
Start:	TocTopic	mydoc.htm	12.4.4
Overwrite any plugin.xml	WritePlugin=Yes	Yes	12.4
Overwrite any contexts.xml	WriteContext=Yes	Yes	12.7

12.2.3 Deciding where to locate configuration settings

When you set up an Eclipse Help project from within FrameMaker, if configuration file `_m2htmlhelp.ini` is not already present in the project directory, **Mif2Go** creates this file for you; see §3 [Converting a book or document](#) on page 77.

Which configuration file?

To configure Eclipse Help output, add settings to one of the following files, depending on the scope of each setting:

<u>Scope</u>	<u>Configuration file</u>	<u>Location</u>
Current project only	<code>_m2eclipse.ini</code>	Current project directory
All Eclipse Help projects	<code>local_m2eclipse_config.ini</code>	<code>%omsyshome%\m2g\local\config\</code>

See §30.5 [Deciding which configuration file to edit](#) on page 856.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §13 [Converting to HTML/XHTML](#) on page 423
- §18 [Splitting and extracting files](#) on page 585
- §21 [Mapping text formats to HTML/XML](#) on page 645
- §23 [Including graphics in HTML](#) on page 703
- §24 [Converting tables to HTML](#) on page 727

12.2.4 Specifying Eclipse Help output options

To add or change any of the options described in this section, edit configuration file `_m2eclipse.ini`, located in the project directory.

In this section:

- §12.2.4.1 [Specifying a different output file extension](#) on page 405
- §12.2.4.2 [Specifying the target Eclipse version](#) on page 406
- §12.2.4.3 [Choosing whether to generate plugin.xml](#) on page 406

12.2.4.1 Specifying a different output file extension

The default output file extension for Eclipse Help is `.htm`. To change the file extension (for example, to specify `.html` instead):

```
[Setup]
FileSuffix = .html
```

12.2.4.2 Specifying the target Eclipse version

By default, **Mif2Go** generates output for Eclipse version 3.3. To produce output for a prior version of Eclipse:

```
[EclipseHelpOptions]
; EclipseVer = Eclipse version point number, default 3, for 3.3
EclipseVer = 2
```

The default value of `EclipseVer` is 3.

When `EclipseVer > 1`, the `<topic>` elements in `index.xml` have a `label` attribute in addition to the `href` attribute. You might need to produce a version of `index.xml` without `label` attributes for 3.1 users, and one with for 3.2+ users.

When `EclipseVer > 2`, the `<extension>` element in `plugin.xml` for contexts uses the `file` attribute instead of the `name` attribute for the contexts file. If some of your Eclipse Help users have Eclipse 3.2 and some have Eclipse 3.3, you must distribute two different `plugin.xml` files.

12.2.4.3 Choosing whether to generate plugin.xml

By default, **Mif2Go** generates manifest file `plugin.xml` for Eclipse Help. If your workflow does not require generating this file because it is produced by some other process, you can exclude `plugin.xml` from your **Mif2Go** project.

To direct **Mif2Go** *not* to produce `plugin.xml`:

```
[EclipseHelpOptions]
; UsePlugin = Yes (default), or No (never write plugin.xml, use if
; not part of the deliverable system because others provide it)
UsePlugin = No
```

12.2.5 Making sure links work in Eclipse Help

The case of file names is significant on the Eclipse platform, even on Windows systems. To avoid case mismatch between links and the files they reference, in some circumstances you might have to specify the following option:

```
[HTMLOptions]
; MakeFileHrefsLower = No (leave case unchanged) or Yes
MakeFileHrefsLower = Yes
```

`MakeFileHrefsLower` is set to `Yes` in system configuration file `d2htm_config.ini`. If you want **Mif2Go** to leave case alone in hypertext links, you can override this setting in a project or local configuration file.

See §19.2.6 [Forcing link text to lowercase](#) on page 613.

12.2.6 Disabling breadcrumb trails in Eclipse Help

Eclipse Help version 3.3 includes breadcrumbs (trails of links) by default. If you do not want this navigation feature, you can disable it.

To disable breadcrumbs, find the file named:

```
org.eclipse.help.webapp\advanced\breadcrumbs.css
```

and replace its contents with:

```
...help_breadcrumbs {
    display: none;
}
```

Including anchors in TOC entries also disables Eclipse Help native breadcrumbs, but might interfere with other features; see §12.4.4.3 [Enabling mid-topic links from the TOC](#) on page 413.

12.3 Configuring Eclipse Help manifest files

In this section:

- §12.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 407
- §12.3.2 [Specifying Eclipse Help plug-in properties](#) on page 407
- §12.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 408
- §12.3.4 [Configuring the plug-in manifest file for Eclipse Help](#) on page 409

12.3.1 Specifying a Java manifest file for Eclipse Help

By default, **Mif2Go** includes a Java manifest file, `MANIFEST.MF`. To omit this file and use only `plugin.xml` as a manifest:

```
[EclipseHelpOptions]
; UseManifest = Yes (default, required for .jars) or No
UseManifest = No
```

If you intend to package your Eclipse Help files in a `.jar` file, you must keep the default value: `UseManifest=Yes`.

If you intend to package your Eclipse Help files in `doc.zip`, set `UseManifest=No`.

When `UseManifest=Yes`, **Mif2Go** does the following:

- Places title, ID, provider, and product-version properties in `MANIFEST.MF`. When `UseManifest=No`, these properties are included in `plugin.xml` instead; see §12.3.2 [Specifying Eclipse Help plug-in properties](#) on page 407.
- Includes a processing instruction in `plugin.xml`, and sets the plug-in schema version to 3.2 unless you specify a different version; see §12.3.4.2 [Including a processing instruction to validate plugin.xml](#) on page 409. When `UseManifest=No`, the processing instruction is not included in `plugin.xml`.
- Creates a subdirectory named `META-INF` in your Eclipse Help project wrap directory, and moves `MANIFEST.MF` into `META-INF`; see §35.6.1 [Specifying a wrap directory](#) on page 961. When `UseManifest=No`, subdirectory `META-INF` is not created.

12.3.2 Specifying Eclipse Help plug-in properties

The properties described here appear either as attributes of the `<plugin>` element in `plugin.xml` or as values of entries in `MANIFEST.MF`, depending on the setting for `UseManifest` (see §12.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 407):

Table 12-2: Eclipse Help properties in either MANIFEST.MF or plugin.xml

Property	Configuration setting	UseManifest=Yes	UseManifest=No
<i>How/where to specify</i>		MANIFEST.MF entry	<plugin> attribute
Title	PluginName	Bundle-Name	name
ID	PluginID	Bundle-SymbolicName	id
Provider name	PluginProvider	Bundle-Vendor	provider-name
Product version	PluginVer	Bundle-Version	version

Title To specify a title for your Eclipse Help plug-in:


```
[EclipseHelpOptions]
; PluginName = text used in <plugin> or manifest for name attribute
PluginName = Mif2Go Eclipse Help
```

In MANIFEST.MF, this title becomes the Bundle-Name value. The default value is the base name of your FrameMaker book or document.

ID To specify a plug-in ID:

```
[EclipseHelpOptions]
; PluginID = text used in <plugin> or manifest for id attribute
PluginID = com.mif2go.help
```

The plug-in ID identifies your plug-in to other components of the Eclipse Platform. In MANIFEST.MF, this ID becomes the Bundle-SymbolicName value. The default value is com.mif2go.help.

Provider name To specify the plug-in provider:

```
[EclipseHelpOptions]
; PluginProvider = text used in <plugin> or manifest for provider-name
; attribute
PluginProvider = mif2go.com
```

In MANIFEST.MF, the provider name becomes the Bundle-Vendor value. The default value is mif2go.com.

Product version To specify the version of the product your Eclipse Help content is about (*not to be confused with the Eclipse version*):

```
[EclipseHelpOptions]
; PluginVer = text used in <plugin> or manifest for version attribute
PluginVer = 1.0
```

In MANIFEST.MF, this version number becomes the Bundle-Version value. The default value of PluginVer is 1.0.

12.3.3 Configuring the Java manifest file for Eclipse Help

When you package Eclipse Help in a .jar file, the .jar file must include a Java manifest, MANIFEST.MF. **Mif2Go** can create this file for you, and does so by default when UseManifest=Yes; see §12.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 407.

To prevent **Mif2Go** from creating MANIFEST.MF when UseManifest=Yes:

```
[EclipseHelpOptions]
; WriteManifest = Yes (default, always write, even if it exists)
; or No (customized, write only if not found)
WriteManifest = No
```

When WriteManifest=Yes, **Mif2Go** creates MANIFEST.MF in the project directory; if this file is already present, **Mif2Go** overwrites it.

When WriteManifest=No, **Mif2Go** creates MANIFEST.MF only if this file is not already present in the project directory; **Mif2Go** does not overwrite an existing MANIFEST.MF. This setting allows you to customize MANIFEST.MF outside of **Mif2Go**, without losing your customizations during subsequent conversions.

To specify that your plug-in must run as a singleton:

```
[EclipseHelpOptions]
; UseSingleton = No (default) or Yes (add "; singleton:=true" after
; PluginID in manifest file)
UseSingleton = Yes
```

When `UseSingleton=Yes`, **Mif2Go** adds “`; singleton:=true`” after the value of `Bundle-SymbolicName` in `MANIFEST.MF`.

12.3.4 Configuring the plug-in manifest file for Eclipse Help

The plug-in manifest for your Eclipse Help project, `plugin.xml`, describes how your Eclipse Help plug-in extends the Eclipse Platform, and how its functionality is implemented. By default, **Mif2Go** creates this manifest file based on settings you provide. However, you can exclude creation of `plugin.xml` from your project; see §12.2.4.3 [Choosing whether to generate plugin.xml](#) on page 406.

In this section:

- §12.3.4.1 [Creating plugin.xml for Eclipse Help](#) on page 409
- §12.3.4.2 [Including a processing instruction to validate plugin.xml](#) on page 409
- §12.3.4.3 [Specifying the plug-in schema version for plugin.xml](#) on page 410
- §12.3.4.4 [Specifying Eclipse Help TOC properties in plugin.xml](#) on page 410
- §12.3.4.5 [Specifying Eclipse Help index properties in plugin.xml](#) on page 410
- §12.3.4.6 [Including a runtime element in plugin.xml](#) on page 410
- §12.3.4.7 [Including or excluding full-text search for Eclipse Help](#) on page 411
- §12.3.4.8 [Specifying Eclipse Help CSH properties in plugin.xml](#) on page 411

See also:

- §12.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 408

12.3.4.1 Creating plugin.xml for Eclipse Help

To direct **Mif2Go** to create `plugin.xml`:

```
[EclipseHelpOptions]
; WritePlugin = Yes (default, write plugin.xml) or No (write only
; if not already present, use if customized)
WritePlugin = Yes
```

When `WritePlugin=Yes`, **Mif2Go** creates `plugin.xml` in the project directory; if `plugin.xml` is already present, **Mif2Go** overwrites it.

When `WritePlugin=No`, **Mif2Go** creates `plugin.xml` only if this file is not already present in the project directory; **Mif2Go** does not overwrite an existing `plugin.xml`. This setting allows you to customize `plugin.xml` outside of **Mif2Go**, without losing your customizations during subsequent conversions.

12.3.4.2 Including a processing instruction to validate plugin.xml

By default, **Mif2Go** includes a processing instruction (PI) in `plugin.xml`, specifying the plug-in schema version used to validate `plugin.xml`. To omit the processing instruction:

```
[EclipseHelpOptions]
; IncludeVersionPI = Yes (default, include PI with version
; specified by PluginSchemaVersion at start of plugin.xml) or No
IncludeVersionPI = No
```

When `IncludeVersionPI=Yes`, **Mif2Go** includes a PI of the following form at the start of `plugin.xml`:

```
<?eclipse version="3.2"?>
```

This processing instruction is *required* when you provide Eclipse Help files in a `.jar` file; see §12.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 408. The value of

the version attribute represents the plug-in schema version; see §12.3.4.3 [Specifying the plug-in schema version for plugin.xml](#) on page 410.

12.3.4.3 Specifying the plug-in schema version for plugin.xml

To specify the plug-in schema version (*not to be confused with the plug-in product version or the Eclipse version*):

```
[EclipseHelpOptions]
; PluginSchemaVersion = version used to validate plugin.xml.
PluginSchemaVersion = 3.2
```

The value of `PluginSchemaVersion` becomes the value of the version attribute in a processing instruction (PI) in `plugin.xml`; see §12.3.4.2 [Including a processing instruction to validate plugin.xml](#) on page 409.

If you do not specify a value for `PluginSchemaVersion`, the default value depends on the value of `UseManifest`:

- If `UseManifest=Yes`, `PluginSchemaVersion=3.2`.
- If `UseManifest=No`, `PluginSchemaVersion=3.1`.

If you are providing Eclipse Help in a `.jar` file (see §12.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 408), the default value (in fact *the only valid value*) of `PluginSchemaVersion` is 3.2. Do not set `PluginSchemaVersion` to 3.3, even if `EclipseVer=3` (see §12.2.3 [Deciding where to locate configuration settings](#) on page 405).

If you are providing Eclipse Help in a `.zip` file, the value of `PluginSchemaVersion` can be either 3.0 or 3.1; the default value is 3.1.

12.3.4.4 Specifying Eclipse Help TOC properties in plugin.xml

To specify TOC properties in `plugin.xml`:

```
[EclipseHelpOptions]
; TocFilename = name for contents file, always toc.xml for primary
TocFilename=toc.xml
; TocPrimary = Yes (default, toc.xml) or No (secondary)
TocPrimary=Yes
; TocExtradir = path to dir for additional docs to include
; in search index, even if not referenced from any toc topic
TocExtradir =
```

See §12.4.4 [Configuring contents properties for Eclipse Help](#) on page 412.

12.3.4.5 Specifying Eclipse Help index properties in plugin.xml

To specify index properties in `plugin.xml`:

```
[EclipseHelpOptions]
; IdxFilename = name for index file, normally index.xml
IdxFilename=index.xml
; UseIndex = No (default) or Yes (for newer releases of Eclipse)
UseIndex = Yes
```

See §12.4.5 [Configuring index properties for Eclipse Help](#) on page 414.

12.3.4.6 Including a runtime element in plugin.xml

By default, **Mif2Go** omits the `<runtime/>` element from `plugin.xml`; this element can cause problems in Eclipse 3.6 and later versions. However, the `<runtime/>` element was

required in earlier versions of Eclipse, so whether it should be present depends on the version of the Eclipse platform where your Eclipse Help system will be deployed.

To have **Mif2Go** include a `<runtime/>` element in `plugin.xml`:

```
[EclipseHelpOptions]
UseRuntime = Yes
```

12.3.4.7 Including or excluding full-text search for Eclipse Help

By default, **Mif2Go** includes an extension point for full-text search in `plugin.xml`. To exclude this extension point:

```
[EclipseHelpOptions]
; UseFTS = Yes (default, adds extension point to plugin file) or No
UseFTS = No
```

12.3.4.8 Specifying Eclipse Help CSH properties in plugin.xml

To specify CSH properties used to produce infopops, in `plugin.xml`:

```
[EclipseHelpOptions]
; UseContext = Yes (default, add context ref to plugin.xml) or No
UseContext = Yes
; WriteContext = Yes (default, write contexts.xml) or No
WriteContext = Yes
; ContextDescription = Yes (default, include) or No (omit)
ContextDescription = Yes
; DescriptionIsFirstLabel = No (default) or Yes (use the label from
; the first context item as the description for the context)
DescriptionIsFirstLabel = No
; ContextAnchors = No (default, filename only) or Yes (refer to para)
ContextAnchors = No
```

To set contexts, use **EclipseContext** markers in your FrameMaker document. Place an **EclipseContext** marker that contains a context ID name in each topic to be referenced by an infopop. Add the infopop description in the **Mif2Go** configuration file, in section `[EclipseHelpContexts]`.

See §12.7 [Setting up CSH for Eclipse Help](#) on page 417.

12.4 Configuring contents and index for Eclipse Help

In this section:

- §12.4.1 [Choosing contents and index methods for Eclipse Help](#) on page 411
- §12.4.2 [Supplying path information for contents and index links](#) on page 412
- §12.4.3 [Encoding special characters for contents and index entries](#) on page 412
- §12.4.4 [Configuring contents properties for Eclipse Help](#) on page 412
- §12.4.5 [Configuring index properties for Eclipse Help](#) on page 414

12.4.1 Choosing contents and index methods for Eclipse Help

To direct **Mif2Go** to generate only contents, or both contents and index, for Eclipse Help:

```
[EclipseHelpOptions]
; ListType (for filter to create) = Contents (default)
; or Both (with index, for Eclipse 3.2+)
ListType = Contents
```

When `ListType=Contents` (the default), **Mif2Go** creates only `toc.xml`. Contents file `toc.xml` is required for Eclipse Help.

When `ListType=Both`, **Mif2Go** creates both `toc.xml` and `index.xml`. Index file `index.xml` is supported only in Eclipse version 3.2 and later versions.

You can choose how **Mif2Go** generates data files needed for Eclipse Help contents and index; however, most likely you will never have a reason to change the default setting:

```
[EclipseHelpOptions]
; RefFileType = Full (default) or Body.
RefFileType = Full
```

`RefFileType` values have the following effects:

Full	<i>Default.</i> Mif2Go creates a <code>toc.xml</code> (and an <code>index.xml</code> file, if <code>ListType=Both</code>) for each original <code>FrameMaker</code> file.
Body	Mif2Go creates DCL <code>.bhc</code> (and <code>.bhk</code> , if <code>ListType=Both</code>) files that are merged later to produce <code>toc.xml</code> and <code>index.xml</code> .

See also:

§7.3.4 [Modifying contents or index production for HTML-based Help](#) on page 206.

§12.4.4 [Configuring contents properties for Eclipse Help](#) on page 412

§12.4.5 [Configuring index properties for Eclipse Help](#) on page 414

12.4.2 Supplying path information for contents and index links

If your HTML files will not be in the same directory with `toc.xml` and `idx.xml`, links from these files must include a path.

To supply a path for contents and index links, and also for `<context>` elements (see §12.7 [Setting up CSH for Eclipse Help](#) on page 417):

```
[EclipseHelpOptions]
; TocIdxFilePrefix = prefix to insert at start of file URLs, when
; html files are placed in a subdirectory under the toc and idx
TocIdxFilePrefix = path/to/htmlfiles
```

The path specified by `TocIdxFilePrefix` is relative to the directory where `toc.xml` and `idx.xml` are located, and must designate a subdirectory under that directory.

12.4.3 Encoding special characters for contents and index entries

If entries in your Eclipse Help contents or index show up with special characters represented as numeric entity references (`&#nnn;`), try setting:

```
[HTMLOptions]
ValidOnly = Yes
```

See §21.6.1 [Understanding how Mif2Go represents characters](#) on page 658.

If the results are not satisfactory, try the following settings to force a different method:

```
[HTMLOptions]
Encoding = UTF-8
NumericCharRefs = No
```

See §14.3.3 [Specifying character encoding for generic XML](#) on page 460.

12.4.4 Configuring contents properties for Eclipse Help

Each Eclipse Help system has one primary TOC, and can have any number of secondary (linked) TOCs. The primary TOC file is always named `toc.xml`. A secondary TOC can have any user-defined name with file extension `.xml`. In most cases, the TOC and helpset **Mif2Go** generates will be the primary TOC and helpset.

Properties described in the following sections are included in `toc.xml`:

§12.4.4.1 [Providing a title for the TOC](#) on page 413

§12.4.4.2 [Specifying a starting topic](#) on page 413

§12.4.4.3 [Enabling mid-topic links from the TOC](#) on page 413

§12.4.4.4 [Directing TOC links to top of topic page](#) on page 414

§12.4.4.5 [Avoiding skipped heading levels in the TOC](#) on page 414

Additional TOC properties are specified in the plug-in manifest; see §12.3.4.4 [Specifying Eclipse Help TOC properties in plugin.xml](#) on page 410. See also §7.4 [Configuring contents entries for Help systems](#) on page 209.

12.4.4.1 Providing a title for the TOC

To provide a label to be displayed as the “book” level of the helpset in the Eclipse Help Contents pane:

```
[EclipseHelpOptions]
; TocLabel = text that appears in the Eclipse Help Contents pane
TocLabel = Title for your Eclipse Help system
```

The default value of `TocLabel` is the value supplied for `PluginName`; see §12.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 407. This is not usually what you want.

12.4.4.2 Specifying a starting topic

To specify which topic file to display when Eclipse Help opens:

```
[EclipseHelpOptions]
; TocTopic = name of opening topic file (required)
TocTopic = startingtopic.htm
```

The default starting topic is the first HTML file listed in the generated contents. That is, if you do not include *any* setting for `TocTopic`, **Mif2Go** generates an entry of the following form in `toc.xml`:

```
<toc label="Title of your Help system" topic="firsttopiclisted.htm">
```

When an explicit file is named as a `topic` attribute, Eclipse does not generate a default help page. To allow Eclipse (at least later versions) to generate a default help page, you can avoid specifying an opening topic by giving `TocTopic` an empty value:

```
[EclipseHelpOptions]
TocTopic =
```

When you specify an empty value, **Mif2Go** omits the `topic` attribute and generates an entry of the following form in `toc.xml`:

```
<toc label="Title of your Help system">
```

In this case, Eclipse generates a default opening page with the following content:

```
Title of your Help system
Contents
Link to the first top-level topic
Link to the second top-level topic
. . .
Link to the last top-level topic
```

12.4.4.3 Enabling mid-topic links from the TOC

The presence of anchors in TOC entries (such as `href="topic.htm#anchor"`) breaks Eclipse Help native breadcrumbs (Eclipse bug 184787), and possibly more. Therefore, by

default **Mif2Go** omits these anchors. However, omitting the anchors also prevents mid-topic jumps from the TOC.

To enable mid-topic jumps by including anchors in Eclipse Help TOC entries:

```
[EclipseHelpOptions]
; TocNamesFileOnly = Yes (default, workaround for Eclipse bug 184787,
; and others, where use of #aname breaks Eclipse native breadcrumbs),
; or No (allows direct midtopic jumps to points within files).
TocNamesFileOnly = No
```

When `TocNamesFileOnly=Yes` (the default), **Mif2Go** omits anchors from TOC entries, enabling Eclipse Help native breadcrumbs but disabling mid-topic jumps from the TOC.

When `TocNamesFileOnly=No`, **Mif2Go** includes anchors in TOC entries, enabling mid-topic jumps from the TOC, but disabling Eclipse Help native breadcrumbs.

See also:

§12.2.6 [Disabling breadcrumb trails in Eclipse Help](#) on page 406

12.4.4.4 Directing TOC links to top of topic page

To make sure links from `toc.xml` go to top-of-page, so that any navigation links you have positioned above the topic heading are visible when a user clicks a TOC link, assign the following property to each paragraph format that is a target of a TOC link:

```
[HTMLParaStyles]
TopicHeading = NoRef
```

See §19.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 614.

12.4.4.5 Avoiding skipped heading levels in the TOC

If you find that Eclipse Help output includes `<topic>` elements with no label attributes, check whether there are any skipped heading levels in your document. Hardly any Help systems allow skipped levels in the TOC. This usually happens when there are sub-subheadings directly under a major heading, before the first subheading. For example:

```
1. Top-level heading
   Third-level heading
   1.1 Second-level heading
       Third-level heading
   1.2 Second-level heading
```

By default, **Mif2Go** inserts the missing level; this works for some systems, but not for Eclipse Help. Avoid this pattern in your document; at the least, exclude such sub-subheadings from the TOC.

12.4.5 Configuring index properties for Eclipse Help

Only Eclipse version 3.2 and later versions support an index. **Mif2Go** produces the index from your FrameMaker index markers, in the form of XML file `index.xml` with links to your HTML topic files. When `EclipseVer=2` or later (see §12.2.3 [Deciding where to locate configuration settings](#) on page 405), **Mif2Go** includes a `label` attribute in each index entry.

At present the only configuration settings that apply to an Eclipse Help index are those you can specify for the plug-in manifest file; see §12.3.4.5 [Specifying Eclipse Help index properties in plugin.xml](#) on page 410. See also §7.5 [Configuring index entries for Help systems](#) on page 211.

If you see style tags in your generated `index.xml` file, most likely they represent formats that are not present in your FrameMaker catalog; the remedy is to catalog those formats, or replace them by catalogued formats in your FrameMaker document. **Mif2Go** leaves them in the output as a diagnostic.

12.5 Configuring search properties for Eclipse Help

For Eclipse version 3.4 or later, to prevent Eclipse from including in each search result the first 170 or so characters after the `<body>` tag of the topic, provide a `<meta>` element containing a `description` attribute in the `<head>` section of the topic; for example:

```
<meta name="description" content="How to polish widgets." />
```

You can specify the `content` value in a FrameMaker marker of type **MetaDescription**, or use the content of a dedicated paragraph format; see §13.4.6 [Supplying content for the `<meta>` tag](#) on page 434.

Note: In Eclipse 3.4 this technique works only for HTML files, not XHTML files.

12.6 Merging Eclipse Help projects

If your customers are using Eclipse version 3.4 or later, you can simply insert marker in FrameMaker for links to secondary TOCs from the primary content. The secondary modules do not need to know where they are used, and if any are missing, the primary helpset still works without error. The missing TOC items are silently omitted. See §12.6.1 [Linking primary content to secondary TOCs](#) on page 415.

If you must support versions of Eclipse prior to Eclipse 3.4, you might have to use anchors in the primary TOC and links to those anchors from any secondary modules; see §12.6.2 [Linking secondary TOCs to primary content \(deprecated\)](#) on page 416. For versions of Eclipse starting with Eclipse 3.4, this method is deprecated.

In this section:

§12.6.1 [Linking primary content to secondary TOCs](#) on page 415

§12.6.2 [Linking secondary TOCs to primary content \(deprecated\)](#) on page 416

12.6.1 Linking primary content to secondary TOCs

To link a secondary helpset into your primary Eclipse Help system, insert a marker of type **EclipseLink** in the FrameMaker document to be converted to your primary helpset, in the paragraph that immediately precedes the point where you want the secondary helpset to appear. **EclipseLink** marker content consists of two items separated by a space:

- TOC level number
- Path to the secondary TOC file.

The TOC level number is an integer that corresponds to whatever level number you specified for primary-TOC entries at the same level; see §7.4.4 [Setting contents levels for HTML-based Help](#) on page 210.

In Eclipse Help output, **Mif2Go** generates the following from each **EclipseLink** marker:

```
<link toc="path/to/secondary/toc.xml">
```

If you do not know ahead of time which secondary helpset (if any) will be needed at a given point in a primary helpset, insert an **EclipseLink** marker for each possible secondary. In Eclipse version 3.4 and later versions, those `<link toc="...">` elements that specify paths to helpsets that are not present at build time are ignored. For Eclipse Help versions

earlier than 3.4, see §12.6.2 [Linking secondary TOCs to primary content \(deprecated\)](#) on page 416.

12.6.2 Linking secondary TOCs to primary content (*deprecated*)

The methods described in this section apply to Eclipse version 3.3 and earlier versions. For Eclipse version 3.4 and later versions, see §12.6.1 [Linking primary content to secondary TOCs](#) on page 415 instead.

To link a secondary TOC to an anchor in the primary content (or in another secondary TOC):

```
[EclipseHelpOptions]
; TocLinkTo = path to another (secondary) TOC with anchor,
; such as ../anotherPlugin/api.xml#moreapi, for link_to attribute
TocLinkTo = ../path/to/anotherPlugin/otherTOC.xml#moreinfo
```

The value of `TocLinkTo` is used for a `link_to` attribute in the secondary TOC. The secondary TOC file must have a name other than `toc.xml`.

To specify where in the primary helpset a secondary TOC should appear, in your FrameMaker document insert an **EclipseAnchor** marker in the paragraph that immediately precedes the point where you want the secondary helpset linked.

You can use a marker either of type **EclipseAnchor** or of type **EclipseLink**. Marker content consists of the following:

EclipseAnchor	TOC level number, space, anchor name.
EclipseLink	TOC level number, space, path to the secondary TOC file.

The TOC level number is an integer that corresponds to whatever level number you specified for primary-TOC entries at the same level; see §7.4.4 [Setting contents levels for HTML-based Help](#) on page 210. The anchor name provides a target for a link from a secondary TOC.

Which marker type you use depends on which scenario you anticipate:

[Alternative or optional secondary TOCs](#)
[Alternative primary TOCs.](#)

*Alternative or
optional
secondary TOCs*

If you do not know ahead of time which sub-module (if any) will be needed at a given point in a primary helpset, use an **EclipseAnchor** marker in the primary system. Each sub-module `subTOC.xml` must include the anchor name in its `link_to` attribute.

*EclipseAnchor
rationale*

Suppose you are creating help systems to be deployed with Eclipse version 3.3 or earlier, and you do not know which modules any given customer has, so you ship separate modules to be merged. If you specify all possible modules in the primary helpset, using **EclipseLink**, the customer gets broken links for any missing modules. So instead, you use an **EclipseAnchor** in the primary helpset for each sub-module; the marker content is not rendered, but it tells the sub-module where to hook in. When the sub-modules are alternatives, you do not need to rebuild the primary system even when you create more alternatives; just use the existing anchor. You cannot do that with **EclipseLink**; you would have to rebuild the primary helpset with an added link every time you created a new alternative sub-module.

*Alternative
primary TOCs*

If you do not know ahead of time into which system a given sub-module must fit, insert an **EclipseLink** marker in each of the alternative primary systems, specifying the path to the sub-module.

*EclipseLink
rationale*

Suppose you have many standard components, but a new framework for each customer. Then it might seem more direct to use **EclipseLink**, and link from the primary helpset to the secondary explicitly.

12.7 Setting up CSH for Eclipse Help

Infopops serve the purpose of Context-Sensitive Help for Eclipse Help. Infopops are intended to contain only a sentence or two of descriptive information, plus one or more hypertext links pointing to further information.

In this section:

- §12.7.1 [Understanding how Mif2Go generates context links](#) on page 417
- §12.7.2 [Naming context file and attribute for secondary plug-ins](#) on page 417
- §12.7.3 [Configuring context IDs and context anchors](#) on page 418
- §12.7.4 [Configuring context descriptions](#) on page 418
- §12.7.5 [Locating context information](#) on page 419

See also:

- §7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239

12.7.1 Understanding how Mif2Go generates context links

Mif2Go recognizes custom FrameMaker **EclipseContext** markers as the targets of infopop calls. Each infopop provides a link to the topic where an **EclipseContext** marker is inserted. The `<topic>` elements included get their `href` and `label` attributes from the `<topic>` elements of the containing paragraphs. This provides the equivalent of the aliases used for CSH in other Help formats; see §7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239. For example:

In `plugin.xml`:

```
<extension point="org.eclipse.help.contexts">
  <contexts file="contexts.xml"/>
</extension>
```

Note: Eclipse 3.1 and 3.2 require a *name* parameter for the `<contexts>` element in `plugin.xml`; Eclipse 3.3 requires a *file* parameter instead. See §12.2.3 [Deciding where to locate configuration settings](#) on page 405.

In `contexts.xml`:

```
<contexts>
  <context id="help_button">
    <description>Brief description of this control.</description>
    <topic href="file_name_link1.html" label="Link1 Topic Title"/>
    <topic href="file_name_link2.html" label="Link2 Topic Title"/>
  </context>
  . . .
</contexts>
```

Mif2Go creates `contexts.xml` afresh every time you run the conversion, unless you say not to; see §12.3.4.8 [Specifying Eclipse Help CSH properties in plugin.xml](#) on page 411.

12.7.2 Naming context file and attribute for secondary plug-ins

To specify a name for the context file:

```
[EclipseHelpOptions]
; ContextFileName = name for context file, default contexts.xml
ContextFileName=contexts.xml
```

The default name of the context file is `contexts.xml`.

To specify a plug-in attribute for secondary plug-ins:

```
[EclipseHelpOptions]
; ContextPluginName = plug-in attribute for secondary plugins,
; default none
ContextPluginName=
```

By default, no attribute is provided for a secondary plug-in.

12.7.3 Configuring context IDs and context anchors

To provide entry points for infopop calls from an application to an Eclipse Help system, insert **EclipseContext** markers in target topics in your FrameMaker file. The content of an **EclipseContext** marker is the context ID for the infopop, which becomes a `<context>` element in `toc.xml`.

By default, **Mif2Go** inserts each context anchor at the beginning of a topic, regardless of where in the topic you place the **EclipseContext** marker in FrameMaker. To produce mid-topic entry points instead:

```
[HTMLOptions]
ObjectIDs = All

[EclipseHelpOptions]
; ContextAnchors = No (default, filename only) or Yes (refer to para)
ContextAnchors = Yes
```

When `ContextAnchors=Yes`, **Mif2Go** inserts a context anchor at the beginning of the paragraph that contains an **EclipseContext** marker.

When `ContextAnchors=No`, the context anchor appears at the beginning of the topic.

See also:

§34.1.2 [Using markers to add links and instructions](#) on page 935

12.7.4 Configuring context descriptions

By default, **Mif2Go** includes a `<description>` element for each `<context>` element in `toc.xml`.

To provide content for the `<description>` element:

```
[EclipseHelpContexts]
; ContextID = short plain-text description for its infopop
```

For example:

```
[EclipseHelpContexts]
ChooseProject = Choose a project from the list, or name a new project.
Export = Choose final options and make last-minute changes.
```

To have **Mif2Go** copy the `<description>` value from the `<topic label>` value instead:

```
[EclipseHelpOptions]
; DescriptionIsFirstLabel = No (default) or Yes (use the label from
; the first context item as the description for the context)
DescriptionIsFirstLabel = Yes
```

To omit the `<description>` element:

```
[EclipseHelpOptions]
; ContextDescription = Yes (default, include) or No (omit)
ContextDescription = No
```

12.7.5 Locating context information

If the HTML files in your project will not reside in the same directory as `toc.xml`, you must provide path information for the `<context>` elements; see §12.4.2 [Supplying path information for contents and index links](#) on page 412.

12.8 Packaging Eclipse Help files

Eclipse Help allows you to deploy your HTML topic files (but not the XML files) in a ZIP file called `doc.zip`, or XML and HTML files in a JAR file called `doc.jar`. Although packaging for topic files is not required for Eclipse Help, it is recommended.

Mif2Go provides ZIP packaging, if you provide a ZIP program (such as `pkzip.exe` or WinZip command-line add-on `wzip.exe`) and specify the command and parameters required to execute the program.

Mif2Go can provide JAR packaging, if you provide the Java `jar.exe` program and its environment variables, and specify the `jar` command and parameters.

In this section:

- §12.8.1 [Specifying a ZIP command for doc.zip](#) on page 419
- §12.8.2 [Specifying ZIP command parameters](#) on page 419
- §12.8.3 [Specifying a JAR command for doc.jar](#) on page 420
- §12.8.4 [Monitoring the packaging step for errors](#) on page 420
- §12.8.5 [Archiving Eclipse Help files](#) on page 420

See also:

- §7.2.4 [Compiling and distributing Help systems](#) on page 204
- §35.6 [Assembling files for distribution](#) on page 961

12.8.1 Specifying a ZIP command for doc.zip

To specify a ZIP command to create `doc.zip`:

```
[EclipseHelpOptions]
; ZipCommand = zip command without parameters
ZipCommand = path/to/zip_program
```

If your ZIP program is not located in a directory that is on the system `PATH`, include a full absolute path for `ZipCommand`. If the path includes spaces, enclose the entire path in quotes. For example, for `wzip`:

```
[EclipseHelpOptions]
ZipCommand = "c:/program files/winzip/wzip"
```

12.8.2 Specifying ZIP command parameters

To specify parameters to the ZIP command (for example, for `wzip`):

```
[EclipseHelpOptions]
; ZipParams = parameters to issue for ZipCommand
ZipParams = -a doc.zip *.htm *.jpg *.gif *.css
```

For parameters that are to be passed to the ZIP program, observe the following:

- Do not enclose parameter values in quotes.
- Use backslashes as separators in path-name parameters.
- Use a dash (“-”) instead of a forward slash to prefix a command option.

Note: The name of the file created by your ZIP program *must* be `doc.zip`.

If `[Automation]WrapAndShip=Yes` (see §35.2 [Activating and logging production of deliverables](#) on page 956), file specifications you list as parameters for `ZipParams` should be relative to the `[Automation]WrapPath` directory, otherwise to the project directory.

Or, you can specify absolute paths (for example, for `wzzip`):

```
[EclipseHelpOptions]
ZipParams = -a doc.zip -xg:\eh\wrap\*.xml g:\eh\wrap\*.*
```

This `wzzip` parameter list includes everything in directory `g:\eh\wrap` *except* XML files. Do not include `plugin.xml`, `toc.xml`, `index.xml`, or any other Eclipse Help XML files in `doc.zip`. To archive files for storage or shipping, see §12.8.5 [Archiving Eclipse Help files](#) on page 420.

12.8.3 Specifying a JAR command for `doc.jar`

To produce `doc.jar`, use a **Mif2Go** system command. For example:

```
[Automation]
SystemEndCommand = jar -cvf doc.jar *
```

See §34.4.1 [Specifying system commands](#) on page 938.

If you give your JAR file a name that allows it to be included in the `ECLIPSE_HOME/plugins` directory, you do not need to provide a subdirectory for your Eclipse Help system. For example:

```
[Automation]
SystemEndCommand = jar -cvf com.myapp.eclipse.core.doc.jar *
```

When you JAR your Eclipse Help system, you have exactly one deliverable, so you do not need to do any further archiving.

12.8.4 Monitoring the packaging step for errors

You might want to set the following option, so you can see any error messages that result from packaging Eclipse Help components:

```
[Automation]
; KeepCompileWindow = No (default)
; or Yes (so any error messages can be seen)
KeepCompileWindow=Yes
```

When `KeepCompileWindow=Yes`, a system window opens when the packaging command (`zip` or `jar`) runs. If there are no errors, you will see only a command prompt when the process finishes. You must dismiss the window before **Mif2Go** can continue processing.

12.8.5 Archiving Eclipse Help files

If you package your Eclipse Help files via ZIP (or if you do not package them at all), you can use **Mif2Go** automation settings to archive everything needed for your Eclipse Help plug-in: XML files along with HTML files or `doc.zip`. See §35.11 [Archiving deliverables](#) on page 973. The default archive file name is `plugin.zip`.

If you package your Eclipse Help files via JAR, you do not need to do any further archiving.

13 Converting to HTML/XHTML

This section shows how to set options in your HTML project configuration file. Unless otherwise indicated, settings for HTML apply also to XHTML and to HTML-based Help systems. Topics include:

- §13.1 [Deciding which type of output to produce](#) on page 424
- §13.2 [Setting up an HTML project](#) on page 424
- §13.3 [Including starting code and entity references](#) on page 429
- §13.4 [Supplying values for the <head> element](#) on page 429
- §13.5 [Specifying HTML <body> attributes](#) on page 436
- §13.6 [Specifying document-wide properties for HTML](#) on page 436
- §13.7 [Defining and mapping colors for HTML](#) on page 438
- §13.7 [Defining and mapping colors for HTML](#) on page 438
- §13.8 [Converting generated files for HTML](#) on page 441
- §13.9 [Importing HTML files as insets](#) on page 446
- §13.10 [Converting conditions to HTML attributes](#) on page 446
- §13.11 [Providing hover text for terms in HTML](#) on page 448
- §13.12 [Generating XHTML for Confluence 4.x](#) on page 449
- §13.13 [Exporting content for database input](#) on page 450
- §13.14 [Using framesets](#) on page 450
- §13.15 [Adding a “Made with Mif2Go” label or button](#) on page 452
- §13.16 [Passing W3C validation tests](#) on page 453

See also:

<i>HTML-based help</i>	§7 Producing on-line Help on page 199, if you plan to generate a Help system.
<i>Generic XML</i>	§14 Converting to generic XML on page 457, for settings specific to XML (but <i>not</i> DITA or DocBook XML).
<i>DITA XML</i>	§15 Converting to DITA XML on page 473.
<i>DocBook XML</i>	§17 Converting to DocBook XML on page 557.
<i>File splitting</i>	§18 Splitting and extracting files on page 585 for settings that govern the subdivision of a FrameMaker document into HTML topic files.
<i>Links</i>	§19 Creating HTML links on page 609 and §20 Providing navigation in HTML on page 627 for ways to create navigation links.
<i>Formats</i>	§21 Mapping text formats to HTML/XML on page 645 for settings that map paragraph and character formats to HTML elements, and that position graphics and equations.
<i>CSS</i>	§22 Setting up CSS for HTML on page 681 if you plan to use CSS.
<i>Graphics</i>	§23 Including graphics in HTML on page 703 for ways to convert graphics and equations, and specify image properties.
<i>Tables</i>	§24 Converting tables to HTML on page 727 for ways to specify table structure and display properties.
<i>WAI markup</i>	§25 Generating WAI markup for HTML on page 755 for ways to add WAI (Web Accessibility Initiative) attributes to tables, images, and links.
<i>Macros</i>	§28 Working with macros on page 787 for ways to use Mif2Go macros.

Markers for
markup

§29 [Working with FrameMaker markers](#) on page 831 for ways to use FrameMaker markers to include HTML code and **Mif2Go** directives in your FrameMaker document.

13.1 Deciding which type of output to produce

If you can choose between HTML 4.01 and XHTML 1.0, consider XHTML. If you might eventually move to XML, the *XHTML 1.0 Recommendation* is a good way to make a transition into that area:

<http://www.w3.org/TR/xhtml1/>

According to the W3C, XHTML 1.0 “defines an XML serialization for HTML 4”. Also, HTML 5 uses XML syntax; see:

<http://dev.w3.org/html5/html4-differences/Overview.html>

For HTML 5 output, you will be concerned mainly with providing an appropriate value for DOCTYPE; see §13.4.1 [Specifying HTML/XML version, DOCTYPE, and DTD](#) on page 429.

Unless otherwise indicated, settings for HTML apply also to XHTML, to XML, and to HTML-based Help systems.

Electronic books

If your output is destined for electronic books, XHTML provides input to third-party ePub production tools. The ePub format is basically XHTML with some icing. To produce ePub, you can use **Mif2Go** XHTML output as input to Calibre, which is free:

<http://calibre-ebook.com/>

To produce a single XHTML output file from a FrameMaker book for input to ePub, see §2.5.6 [Producing a single output file from a FrameMaker book](#) on page 73. To provide a TOC for ePub, you can convert your FrameMaker TOC to XHTML; see §5.5.1 [Converting FrameMaker TOC and IX files](#) on page 124.

Internet browsers

If your output will be displayed on the Web, consider the differences among browsers. If you use CSS (see §22 [Setting up CSS for HTML](#) on page 681), some browsers, such as Mozilla, do not display XHTML output properly on the Web; they ignore your CSS files. However, these browsers properly display the same XHTML output viewed locally, and properly display standard HTML output both locally and on the Web.

13.2 Setting up an HTML project

Generating HTML with **Mif2Go** is an entirely different process from using “Save as HTML” in FrameMaker. **Mif2Go** does not use the FrameMaker reference-page HTML mapping tables, nor create an HTML output file from a FrameMaker book file. However, **Mif2Go** does support inclusion of arbitrary JavaScript anywhere in HTML output.

Although you can specify some HTML or XML options within FrameMaker via **Mif2Go** *Set Up* and *Export* dialogs, you have to edit configuration files to specify most options; see §4.1 [Working with Mif2Go configuration files](#) on page 91.

To add or change HTML options, edit the project configuration file appropriate for the output type, located in the project directory:

<u>Output type</u>	<u>Project configuration file</u>
Standard HTML	_m2html_config.ini
XHTML	_m2xhtml_config.ini

Or, to apply the changes to all of your HTML (or XHTML) projects, edit the corresponding configuration template:

```
%omsyshome%\m2g\local\config\local_m2*ml_config.ini
```

See §30.5 [Deciding which configuration file to edit](#) on page 856.

In this section:

- §13.2.1 [Creating an HTML or XHTML project](#) on page 425
- §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425
- §13.2.3 [Preparing a document for conversion to HTML or XHTML](#) on page 426
- §13.2.4 [Specifying a different output file extension](#) on page 427
- §13.2.5 [Checking automatic settings for HTML or XML split files](#) on page 427
- §13.2.6 [Establishing a conversion workflow for HTML](#) on page 427
- §13.2.7 [Checking HTML output files for broken links](#) on page 428
- §13.2.8 [Checking HTML or XML output files for Mif2Go version](#) on page 428
- §13.2.9 [Using XHTML tagging rules for HTML](#) on page 428

13.2.1 Creating an HTML or XHTML project

To create an HTML or XHTML project:

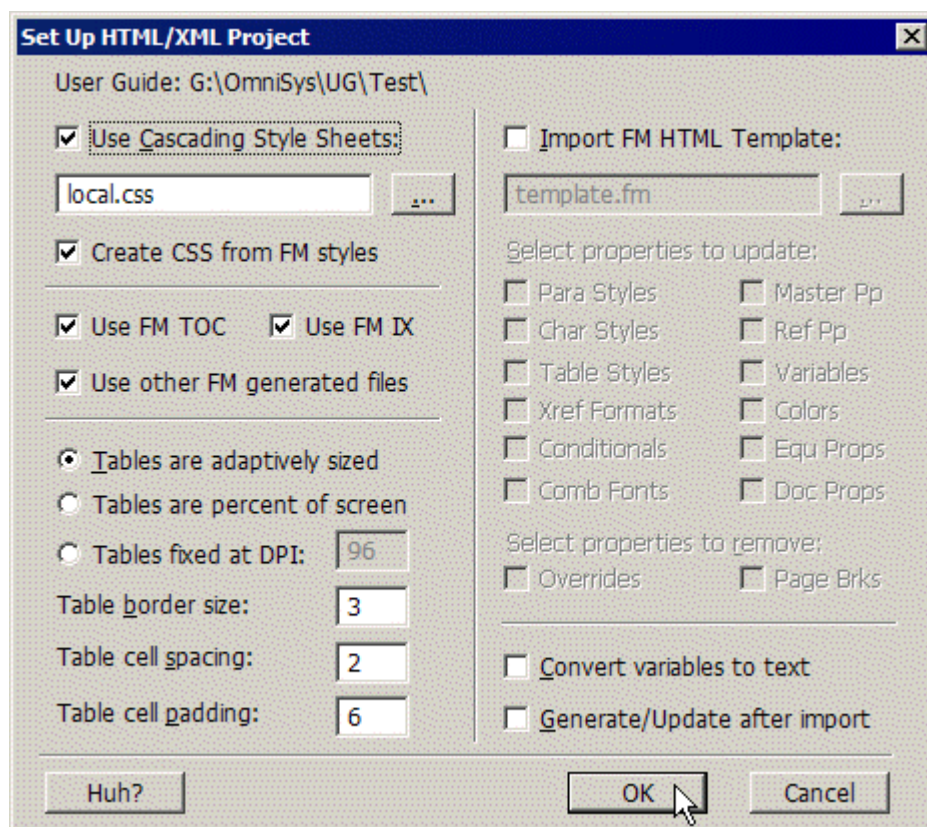
1. Create a directory for output files, separate from the directory where your FrameMaker document is located.
2. Copy the appropriate configuration file `_m2*ml.ini` from `%OMSYSHOME%\m2g\system\starts`, or from an existing **Mif2Go-to-HTML** project, to your newly created project directory.
3. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens.
4. Name your project, and browse to the project directory you created in [Step 1](#) (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
5. Choose one of the following output types:
 - Standard HTML
 - XHTML
6. Check options in the *Set Up HTML/XML Project* dialog (see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425).
7. Use a text editor to edit the resulting configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).
8. **Important:** To make the configuration fit your usage of headings to start topics, pay special attention to section `[HTMLParaStyles]` (see §18.2 [Splitting files](#) on page 586).

13.2.2 Choosing set-up options for an HTML or XHTML project

When you select Standard HTML or XHTML as the output type for a new project, the *Set Up* dialog shown in [Figure 13-1](#) opens. [Table 13-1](#) shows the corresponding settings in the configuration file. *You must edit the configuration file to specify additional options.*

See also:

- §3.4 [Choosing project set-up options](#) on page 79

Figure 13-1 Set Up HTML/XML Project**Table 13-1 HTML and XHTML set-up options and configuration settings**

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Use Cascading Style Sheets	[CSS]	UseCSS=Yes	Yes	22.4
<i>Path to CSS file</i>	[CSS]	CssFileName=mycss.css	local.css	22.4.3
Create CSS from FM styles	[CSS]	WriteCssStylesheet=Once	Once	22.4.3
Tables are adaptively sized	[Tables]	TableSizing=Adaptive	Adaptive	24.4.8
Tables are pct of screen	[Tables]	TableSizing=Percent	Adaptive	24.4.8
Tables fixed at DPI	[Tables]	TableSizing=Fixed	Adaptive	24.4.8
		TableDPI=n	96	
Table border size	[Tables]	Border=n	3	24.4.8
Table cell spacing	[Tables]	Spacing=n	2	24.4.8
Table cell padding	[Tables]	Padding=n	6	24.4.8

13.2.3 Preparing a document for conversion to HTML or XHTML

To successfully convert a FrameMaker document to HTML or XML, observe the following guidelines:

- Use a conversion template that removes page numbers from cross references; see §2.4 [Importing formats from a conversion template](#) on page 67.
- Avoid using tabs; a FrameMaker tab converts to a space in HTML.
- Avoid rotated cells in tables.

- Use referenced graphics rather than embedded graphics where possible.
- Provide versions of graphics in JPEG, GIF, or PNG format; see §5.7 [Processing graphics](#) on page 126 for more information.
- If you are converting a structured document, use distinct formats for distinct presentation effects; see §5.8 [Converting structured documents](#) on page 135.

13.2.4 Specifying a different output file extension

To change the output file extension for HTML or XHTML:

```
[Setup]
FileSuffix = .ext
```

For DITA or DocBook output, see:

§15.2.3 [Specifying DITA output options](#) on page 480

§17.2.3.1 [Changing the DocBook output file extension](#) on page 561

13.2.5 Checking automatic settings for HTML or XML split files

If you are running **Mif2Go** from within FrameMaker, and **Mif2Go** creates a new configuration file, **Mif2Go** tries to determine automatically the most likely headings to start new HTML pages or XML files; see §1.5 [How Mif2Go works](#) on page 62. *You must check these automatic assignments in the configuration file and correct any that are inappropriate*; see §18.2.1 [Designating split points](#) on page 586.

One HTML file per topic is the norm. If you need one monolithic HTML file instead (not recommended), see §18.2.3 [Combining instead of splitting files](#) on page 591.

13.2.6 Establishing a conversion workflow for HTML

Keep all files open For a close-to-WYSIWYG way to set up and run a conversion, you can keep all the following files open at the same time:

- Your document, in FrameMaker.
- Your starting project configuration file, in Notepad (or another text editor).
- The .htm file(s) **Mif2Go** produces, in a browser window.

Refine results All can be visible at once, depending on screen real estate. Now you can do the following, iteratively:

1. Look in the browser window to see something that could be improved.
2. Add or change a line to the configuration file for that improvement.
3. Save the configuration file.
4. On the FrameMaker **File** menu, click **Save Using Mif2Go....**
5. On the browser menu, click **Reload**.

You make all changes either in FrameMaker, without affecting print appearance, or in the configuration file.

If you are converting a large document with many topics and many cross references, deleting previous output files and **Mif2Go**-generated reference (.ref and .grx) files (but *not* configuration or project files) from the project directory between conversions can speed up the conversion process. **Mif2Go** can delete these files for you; see §35.4 [Clearing out old files before converting](#) on page 957. However, do not delete these files if they are needed by other conversions or for other purposes; see §C.5 [Working with reference files for HTML or XML](#) on page 1027.

See also:

§34 [Automating Mif2Go conversions](#) on page 933

§35 [Producing deliverable results](#) on page 955

13.2.7 Checking HTML output files for broken links

When you convert a FrameMaker book to HTML or XML, **Mif2Go** can check for broken interfile links:

```
[Setup]
; CheckLinks = No (default) or Yes (check links after running a book)
CheckLinks=Yes
; CheckLinkLog = D:\path\to\LinkLog.fm to make copy of Book Error Log
CheckLinkLog = path\to\MyBadLinks.fm
; LinkLogAlways = Yes (default) or No (do not display Book Error Log
; if no broken links are found)
LinkLogAlways=Yes
```

See §5.1.5 [Checking for broken links in HTML or XML output](#) on page 112.

13.2.8 Checking HTML or XML output files for Mif2Go version

If you recently installed a **Mif2Go** upgrade or beta version, after you run **Mif2Go**, check to make sure the latest version was actually used to produce HTML output. Windows sometimes caches DLLs, and does not always use a newly replaced DLL until after the system is rebooted.

Open an HTML output file and look at the fourth line. You should see something like the following:

```
<!-- generated by DCL filter dwhtm, Ver 4.0 m193 h272a -->
```

The last two entries identify the build numbers of the **Mif2Go** `drmf.dll` and `dwhtm.dll` components that were used to create the HTML file. See §D.2.9 [Check your version of Mif2Go](#) on page 1034.

13.2.9 Using XHTML tagging rules for HTML

Even if you are creating standard HTML, consider using XHTML tagging. These are the main points to remember:

- Use lowercase for element and attribute names.
- Enclose all attribute values in double quotes.
- Explicitly close all tags.
- Include a space after an element name, even in a closing tag (such as `
`).

All current HTML browsers accept these rules.

When you specify XHTML as your output type, in addition to a name attribute for anchors, **Mif2Go** provides an `id` attribute; for example:

```
<h3><a id="b2d" name="b2d"></a>B2D</h3>
```

This is because XML expects an `id` attribute for many purposes that are handled by the name attribute in HTML. The only way to suppress the `id` attribute is to specify HTML instead of XHTML as the output type.

13.3 Including starting code and entity references

You can specify macro code to be inserted at the very beginning of each HTML output file, and entity references to be inserted before the `<head>` element:

```
[Inserts]
; location = macro to insert, can call another macro
; BeginFile is placed at the very start of the file
; Entities is placed before the HEAD element
```

See §28 [Working with macros](#) on page 787.

To specify an entity reference, create a macro with the entity reference code as the body of the macro, and indicate that the macro is to be placed before the `<head>` element. For example:

```
[Inserts]
Entities=<$MyEntities>

[MyEntities]
<!ENTITY % HTMLlat1 PUBLIC "-//W3C//ENTITIES Latin 1//EN//HTML">
%HTMLlat1;
<!ENTITY % HTMLsymbol PUBLIC "-//W3C//ENTITIES Symbols//EN//HTML">
%HTMLsymbol;
```

Entity references are placed before the `<head>` element in each HTML output file, including split and extracted files.

13.4 Supplying values for the `<head>` element

Mif2Go normally provides a header that indicates compliance with W3C's HTML 4 specification. The default header includes a “transitional” qualifier that permits use of some formatting code that in HTML 4 is deprecated in favor of CSS. Unfortunately, browsers have not quite managed yet to implement CSS well enough so that you can depend on CSS alone; not even CSS1, let alone CSS2. So “strict” compliance has to wait.

In this section:

- §13.4.1 [Specifying HTML/XML version, DOCTYPE, and DTD](#) on page 429
- §13.4.2 [Specifying namespace and language](#) on page 430
- §13.4.3 [Specifying character encoding for HTML](#) on page 431
- §13.4.4 [Including or omitting HTML/XML generator information](#) on page 433
- §13.4.5 [Specifying page titles for HTML output files](#) on page 433
- §13.4.6 [Supplying content for the `<meta>` tag](#) on page 434
- §13.4.7 [Specifying nonstandard values for declarations](#) on page 435

13.4.1 Specifying HTML/XML version, DOCTYPE, and DTD

You can change the HTML or XML header, perhaps to accommodate noncompliant code you are using in a macro, or to conform to the requirements of third-party tools:

```
[HTMLOptions]
; HTMLVersion = version used: 4 (default), 3 (JavaHelp), or 2 (old)
HTMLVersion=4
; UseDOCTYPE = Yes (default) or No (when writing DocBook entity files)
UseDOCTYPE=Yes
; HTMLDocType, PUBLIC identifier required at start of HTML documents
; Default for v4 is: "-//W3C//DTD HTML 4.01 Transitional//EN"
; or if frameset is: "-//W3C//DTD HTML 4.01 Frameset//EN"
; Default for v3 is: "-//W3C//DTD HTML 3.2 Final//EN"
```

```

; Default for v2 is: "-//IETF//DTD HTML 2.0//EN"
; Default for XHTML is: "-//W3C//DTD XHTML 1.0 Transitional//EN"
; Uncomment and give alternate if needed;
; do not leave blank uncommented:
;HTMLDocType="-//W3C//DTD HTML 4.01 Transitional//EN"
; HTMLDTD, the optional SYSTEM identifier in <!DOCTYPE>;
; default is to omit. If you want to add it back, although it breaks
; CSS usage, for v4 it is:
; "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd"
; or for v4 frameset is:
; "http://www.w3.org/TR/1999/REC-html401-19991224/frameset.dtd"
; For XHTML, it is:
; "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"
; Uncomment and leave blank for no DTD (v2 and v3),
; or give an alternate
;HTMLDTD=

```

Mif2Go generates code that is completely compliant with the W3C DTD cited, for whatever version you select. However, it is your responsibility to use only valid syntax for that version in your macros.

HTML 5 For HTML 5 output, leave HTMLVersion=4, and set the other values in this section as appropriate.

JavaHelp When you specify JavaHelp or Oracle Help for Java as the output type, **Mif2Go** automatically sets [HTMLOptions]HTMLVersion=3, unless you override this setting in the configuration file.

XHTML When you specify XHTML as the output type, **Mif2Go** automatically sets the corresponding HTMLDocType and HTMLDTD, unless you override these settings in the configuration file:

```

[HTMLOptions]
HTMLDocType="-//W3C//DTD XHTML 1.0 Transitional//EN"
HTMLDTD="DTD/xhtml11-transitional.dtd"

```

DITA XML When you specify DITA XML as the output type, **Mif2Go** sets HTMLDocType and HTMLDTD as follows, depending on the DITA version and topic type. For example, for DITA version 1.1 and topic type concept:

```

[HTMLOptions]
HTMLDocType="-//OASIS//DTD DITA 1.1 Concept//EN"
HTMLDTD="docs.oasis-open.org/dita/v1.1/CS01/dtd/concept.dtd"

```

You can override the default values; see §15.3 [Specifying general options for DITA](#) on page 483.

DocBook XML When you specify DocBook XML as the output type, **Mif2Go** sets HTMLDocType and HTMLDTD as follows:

```

[HTMLOptions]
HTMLDocType="-//OASIS//DTD DocBook XML V4.5//EN"
HTMLDTD="www.oasis-open.org/docbook/xml/4.5/docbookx.dtd"

```

13.4.2 Specifying namespace and language

You can specify the namespace and language of the <html> tag:

```

[HTMLOptions]
; XHNamespace default for XHTML 1.0 is "http://www.w3.org/1999/xhtml"
; Uncomment and give alternate if needed, do not leave blank
;XHNamespace=http://www.w3.org/1999/xhtml
; XHLanguage default is "en"
;XHLanguage=en
; XHLangAttr = xml:lang (default, set as needed;

```

```
; DocBook wants just lang)
;XHLangAttr=xml:lang
```

Although **Mif2Go** always produces valid code, it does not attempt to validate the content of your macros. We suggest you validate your document with the W3C's free HTML validator:

<http://validator.w3.org/>

and CSS validator services:

<http://jigsaw.w3.org/css-validator/>

If your document is valid, you are offered a little graphic to include in it. See §13.16 [Passing W3C validation tests](#) on page 453.

13.4.3 Specifying character encoding for HTML

HTML is based on Unicode. FrameMaker version 8 and later versions support Unicode, and so does **Mif2Go**, via UTF-8. **Mif2Go** does not directly support non-Unicode double-byte languages (except for Asian and Cyrillic code pages for HTML Help), nor right-to-left languages such as Hebrew and Arabic.

In this section:

§13.4.3.1 [Using special fonts for non-Western languages](#) on page 431

§13.4.3.2 [Selecting a Windows code page for single-byte character sets](#) on page 431

§13.4.3.3 [Specifying encoding for double-byte characters](#) on page 432

See also:

§13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454

§14.3.3 [Specifying character encoding for generic XML](#) on page 460

§21.5 [Assigning properties to text formats](#) on page 653

13.4.3.1 Using special fonts for non-Western languages

FrameMaker versions earlier than FrameMaker 8 do not support Unicode. If you are not working in a Western language, unless you have FrameMaker version 8 or later, you need to use FrameMaker fonts that have your national subset in the “high ASCII” code points (characters with values greater than 0x7F (decimal 127), constitute the “high ASCII” set). **Mif2Go** must then convert those code-point values to the Unicode values.

13.4.3.2 Selecting a Windows code page for single-byte character sets

According to Microsoft, a *code page* is “an ordered set of characters of a given script in which a numeric index (code-point value) is associated with each character”. **Mif2Go** supports fonts that use the following SBCS (Single Byte Character Set) Windows code pages:

1250	Slovenian and other Eastern European and Central European scripts
1251	Cyrillic
1252	Latin I (standard FrameMaker code page)
1253	Greek
1254	Turkish

To specify a Windows code page for your document:

```
[HTMLOptions]
; Ansi = Windows code page to use for FrameMaker font, default 1252
Ansi=1252
```

Use the `Ansi` option to specify character encoding other than the standard FrameMaker encoding, which uses Windows code page 1252.

Set the code page in a marker

If you need a different encoding for only parts of your document, you can set the code page in a marker of type **Ansi**, or in a marker that is mapped in [MarkerTypes] to **Ansi**. The marker should contain only the code-page number. See §29.2.1 [Identifying dedicated custom marker types](#) on page 832.

FrameMaker 8 uses Unicode, not code pages

If you are using FrameMaker version 8, you are using the Unicode character set internally. Times New Roman, and most other standard Microsoft fonts, support Unicode. However, fonts that use a different character set do not. For example, the font “Caecilia LT CE 55 Roman” uses code page 1250; that is what the CE in its name means. For certain characters such as “small a macron”, which is U+0101, there is *no* mapping in code page 1250; so you would not get “small a macron” in this font, because it is not there.

Your best bet is to stick with Unicode fonts, and not try to use others that are CE, CYR, GRK, and so forth. The code-page system was made to provide access to characters within an 8-bit space, and it required remapping of various groups of Unicode characters to fit into that space. FrameMaker 7 and earlier versions support this remapping, but FrameMaker 8 does not. Even if you have to replace some fonts, it is best to move on.

13.4.3.3 Specifying encoding for double-byte characters

Character encoding determines what method is used to represent double-byte characters in the <body> section of HTML output. To specify encoding or, alternatively, numeric references:

```
[HTMLOptions]
; Encoding = ISO-8859-1 (HTML default, numeric refs),
;   or None (write 0x80-0xFF as single characters)
Encoding=ISO-8859-1
; QuotedEncoding = No (default, W3C usage, required for JavaHelp),
;   or Yes (put encoding in meta tag in single quotes, needed by some
;   older browsers)
QuotedEncoding=No
; NumericCharRefs = Yes (default, always use &#nnn;)
;   or No (use UTF-8 for XML)
NumericCharRefs=Yes
```

For XHTML, the **Mif2Go** default is to claim UTF-8 as the encoding, but to use numeric references of the form `&#nnn;` for all characters that would have to be encoded; this satisfies all browsers. That is, **Mif2Go** does not actually produce any characters with values greater than 127 using the UTF-8 encoding; instead, **Mif2Go** uses entities for such characters, readable under any eight-bit encoding scheme.

For XHTML, you can specify a value for `XMLEncoding` (see §14.3.3 [Specifying character encoding for generic XML](#) on page 460) other than the default UTF-8. If you set `Encoding=UTF-8`, you get real UTF-8 encoding (two characters) in place of the numeric character references. However, you can still force use of numeric references by also setting `NumericCharRefs=Yes`.

While `Encoding=None` is not strictly compliant, this setting can be useful in places like Russia, where almost the entire text would otherwise consist of numeric character references. `Encoding=None` provides a 6:1 reduction in such references.

To direct **Mif2Go** to supply single quotes around the `charset` attribute value, specify `QuotedEncoding=Yes`:

```
<meta http-equiv="Content-type" content="text/html; charset='ISO-8859-1'">
```

The default is not to enclose the value in quotes.

See also:

- §13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454
- §14.3.3 [Specifying character encoding for generic XML](#) on page 460
- §21.5 [Assigning properties to text formats](#) on page 653

13.4.4 Including or omitting HTML/XML generator information

The header of each HTML and XML file generated by **Mif2Go** contains an element identifying the **Mif2Go** program that generated the file. By default this element appears in a comment. For example:

```
<!-- generated by DCL filter dwhtml, Ver 3.0 -->
```

You can put this information in a meta tag instead, with the following setting:

```
[HTMLOptions]
; GeneratorTag = Comment (dwhtml version in comment), Meta (tag),
; or None (omit)
GeneratorTag = Meta
```

The generator information is included for accountability, and for troubleshooting; a programmer working with a file created by **Mif2Go** can identify the software version that produced the file.

You can omit the generator information by specifying `GeneratorTag=None`. However, if you need to send files to Omni Systems for support, our developers absolutely need that information to tell what version of **Mif2Go** created the file. It is also helpful for downstream tool vendors, such as editor providers, to know how a file was created. This is an industry standard practice, and the comment form of `GeneratorTag` should be quite harmless.

13.4.5 Specifying page titles for HTML output files

You can specify page titles for an HTML output files any of these ways:

- [Assign a default or computed title](#)
- [Use a heading paragraph](#)
- [Use a FrameMaker marker](#)
- [Assign title text to the HTML file.](#)

The text you supply becomes the content of the `<title>` tag in the HTML `<head>` element. For more information, see §18.4.2 [Specifying page titles for split or extract files](#) on page 594.

Note: If some of your output files show *Test File from Mif2Go* as the title, this means you did not manage to specify titles for those files.

*Assign a default
or computed title*

To specify a default page title for all output files:

```
[HTMLOptions]
;Title = default title for HTML files,
; overridden by all other settings
;Title=Test File from Mif2Go
Title=My default page title
```

You can assign a macro or macro variable to the `Title` keyword. For example, to use the base name of the FrameMaker source file as the page title for each HTML file derived from that FrameMaker file:

```
[HTMLOptions]
Title=<$$_basefile>
```

See §18.4.2.6 [Assigning a default title](#) on page 597.

*Use a heading
paragraph*

The easiest way to provide HTML page titles is to assign the `Title` property to all paragraph formats (usually headings) that begin new HTML files. For example:

```
[HTMLParaStyles]
; Title uses head as HTML page title, see [Titles]
Heading2=Title
```

With this setting, every HTML output file that begins with a *Heading2* paragraph would have the text of that paragraph for a page title. See §18.4.2.2 [Assigning a title with a paragraph format](#) on page 595.

*Use a
FrameMaker
marker*

You can specify an individual page title with a FrameMaker **Title** marker; the content of the **Title** marker becomes the title of the HTML file generated from the section of your document where you inserted the marker. See §18.4.2.5 [Assigning a title with a marker](#) on page 597.

*Assign title text to
the HTML file*

You can provide arbitrary text for the title by assigning the text to the HTML output file name; for example:

```
[Titles]
; html filename = title, overrides [HTMLParaStyles] Title setting
m2r=DCL MIF2RTF Filter Description
```

You must assign the title text to the internal file name assigned by **Mif2Go**, not to any replacement name you may have specified for a split or extract file. See §18.4.2.4 [Assigning a title with a file name](#) on page 596.

13.4.6 Supplying content for the <meta> tag

Mif2Go supports several ways to provide content for <meta> tags in the <head> element of each HTML page.

In this section:

§13.4.6.1 [Providing meta content with paragraph formats](#) on page 434

§13.4.6.2 [Providing meta content with FrameMaker markers](#) on page 435

§13.4.6.3 [Providing meta content with Mif2Go macros](#) on page 435

13.4.6.1 Providing meta content with paragraph formats

When you assign the `Meta` property and a <meta> tag name attribute to a paragraph format, the text of such a paragraph becomes a <meta> tag content attribute for the specified <meta> tag:

```
[HTMLParaStyles]
ParaFmt = Meta
```

Explicitly assigning the `Meta` property to a paragraph format is optional when you assign a tag name to that format in the following section:

```
[StyleMetaName]
; doc style = name to use for meta tag whose content is the para text
ParaFmt = metaname
```

For example, suppose you use paragraph format *Metakeys* to supply content for the `keywords` attribute:

```
[StyleMetaName]
Metakeys=keywords
```

With this setting, the text of every *Metakeys* paragraph would become content for a “keywords” <meta> tag in the <head> element of the HTML file. For example, if the text

of a *Metakeys* paragraph is “staff, location, reporting, roster”, the <meta> tag would look like this:

```
<meta name="keywords" content="staff, location, reporting, roster">
```

You need a different paragraph format for each <meta> tag. For example, if you want author and source tags, you might define paragraph formats *MetaAuthor* and *MetaSource*, and map them as follows:

```
[HTMLParaStyles]
Meta*=Meta Delete

[StyleMetaName]
MetaAuthor=author
MetaSource=source
```

The `Delete` property prevents the content of these special paragraphs from appearing in body text; see §21.3.12 [Eliminating unwanted paragraphs](#) on page 652.

13.4.6.2 Providing meta content with FrameMaker markers

If you give a custom marker type a name that starts with **Meta**, **Mif2Go** automatically makes the marker text the value of whatever attribute is named by the rest of the marker-type name, and puts the attribute and its value in the generated <meta> tag. This method has two advantages:

- Markers do not require entries in the configuration file.
- Markers do not clutter your paragraph catalog.

For example, **MetaKeywords**:

```
<meta name="Keywords" content="whatever was in the marker(s)" />
```

You must ensure that the content of each marker is valid for the named attribute. The text of a FrameMaker marker is limited to 256 characters. **Mif2Go** gets around that restriction by concatenating all markers for the same attribute in the same HTML file. You can just add more markers of the same type, and continue the content. **Meta*** markers are concatenated into one <meta> tag within the same split or extract file, but not across file boundaries.

13.4.6.3 Providing meta content with Mif2Go macros

You can put <meta> tags directly into the configuration file as macros, and not have them in your FrameMaker document at all. See §18.5.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 599 and §28.1 [Defining and invoking macros](#) on page 787.

13.4.7 Specifying nonstandard values for declarations

You can specify different values for several header fields or declarations; however, unless you know you need nonstandard values, you should not need to change any of these. For some settings, the default values vary based on whether the output type is HTML, XHTML, or XML:

```
[HTMLOptions]
; UseXMLRoot = Yes (default) or No (when writing DocBook entity files)
UseXMLRoot=Yes
; XMLRoot default is "html" for XHTML, or "doc" for generic XML.
XMLRoot=html
; UseHeadAndBody = Yes (HTML/XHTML default)
; or No (generic XML and DITA default)
UseHeadAndBody=Yes
; ContentType = text/html (default for HTML and XHTML)
; or application/xml (default for XML); try not to use text/xml
```



```
; (for interoperability)
ContentType=text/html
```

Content-Type is part of MIME, and is used by document-processing tools. Unless you know exactly what you want and need only a mechanism to specify it, leave this setting alone. For more information, see:

http://www.w3.org/Protocols/rfc1341/4_Content-Type.html

For XHTML, you can suppress the `<?xml...?>` declaration, as required by some browsers:

```
[HTMLOptions]
;UseXMLDeclaration = Yes to start with <?xml...?>, or No to omit
UseXMLDeclaration=No
```

13.5 Specifying HTML <body> attributes

You can use configuration settings to assign values to HTML <body> attributes:

```
[Attributes]
; body= attributename=value
```

Keep all attributes on one line, regardless of line length. For XHTML, all attribute names must be lowercase. For example:

```
[Attributes]
body= bgcolor="#FFFFFF" text="#000080" link="#008020" vlink="#804000"
```

You can insert JavaScript for <body> attributes; for example:

```
[Attributes]
body= onLoad="if (self != top) top.location = self.location"
```

In addition to attributes for the <body> tag, you can use the [Attributes] section to specify attribute values for <table>, <tr>, <td>, <th>, <thead>, <tfoot>, and <tbody> tags; see §24.4.1 [Specifying attributes for all tables](#) on page 736.

13.6 Specifying document-wide properties for HTML

In this section:

- §13.6.1 [Specifying a default DPI setting](#) on page 436
- §13.6.2 [Converting system variables to text for HTML](#) on page 437
- §13.6.3 [Suppressing closing </p> tags for HTML](#) on page 437
- §13.6.4 [Suppressing line breaks in HTML and XML output](#) on page 437
- §13.6.5 [Preventing adjacent <pre> elements from merging](#) on page 438

13.6.1 Specifying a default DPI setting

To convert from other sizes to pixels, for such purposes as table-column sizing, indenting tables, and indenting graphics, **Mif2Go** uses this DPI setting:

```
[HTMLOptions]
; ConversionDPI = 96 (default), used when converting sizes to pixels
ConversionDPI = 96
```

If you imported graphics into FrameMaker at a DPI other than 96, be sure to change this setting to the actual DPI you used for import. See §23.9 [Scaling images for HTML](#) on page 719.

13.6.2 Converting system variables to text for HTML

The only way **Mif2Go** can get the text content of FrameMaker system variables (such as date and time) into HTML output is to convert these variables to text. Other variables are already available in a usable form, in the MIF files.

To convert date/time and file-name system variables *on body pages* to text, so they can appear in HTML output, specify the following setting:

```
[Setup]
; ConvertVariables = No (default) or Yes (convert to plain text)
ConvertVariables=Yes
```

For example, you might want to use this setting to get the value of Creation Date or Modification Date to appear in metadata.

Note: This setting does not apply to system variables *on master pages*. **Mif2Go** does not process master-page variables for HTML output.

Note: For system variables to show up in the MIF files, **Mif2Go** must read your original FrameMaker files. If you specify **Use existing MIF** on the *Export* dialog, or in your project configuration file, system variables are not converted.

13.6.3 Suppressing closing </p> tags for HTML

By default, **Mif2Go** provides closing tags `</tagname>`, to conform to W3C validation requirements for XHTML. (One exception: unless you specify XHTML as the output type, **Mif2Go** does not generate closing tags for list items; see §21.12.2 [Converting list formats to HTML list styles](#) on page 675.)

To eliminate `</p>` closing tags:

```
[HTMLOptions]
; NoParaClose = No (default) or Yes (suppress </p> closing tags)
NoParaClose = Yes
```

13.6.4 Suppressing line breaks in HTML and XML output

By default, **Mif2Go** inserts `\n` line breaks in HTML and XML output in several places, including (but not limited to) the following:

- after each `<a name=` so the tag name always appears as the first item on the next line, to make the names easier to find when you inspect **Mif2Go** output
- at the first space that occurs in a paragraph at or after 70 characters (not counting character tags), to make long paragraphs easier to inspect or edit.

These line breaks do not affect HTML display. However, if you are generating XML to be imported into a system that treats `\n` line breaks as though they were paragraph breaks, you might want to get rid of all unintended line breaks in text. See §14.3.5 [Preventing arbitrary line breaks in XML text elements](#) on page 461.

To suppress `\n` line breaks after `<a name=`:

```
[HTMLOptions]
; ATagLineBreak = Yes (default, \n before first attr) or No
ATagLineBreak=No
```

To suppress `\n` line breaks only in preformatted text:

```
[HTMLOptions]
; UnwrapPRE = No (default) or Yes (ignore line breaks in PRE)
UnwrapPRE = Yes
```

To prevent FrameMaker line wraps from becoming \n line breaks in preformatted text:

```
[HTMLOptions]
; IgnoreWrap = No (default, \n where wrap occurs) or Yes
IgnoreWrap = Yes
```

See §21.10.1 [Eliminating line wraps in preformatted text](#) on page 670.

To suppress \n line breaks in all paragraphs:

```
[HTMLOptions]
; NoWrap = No (default, \n where space occurs) or Yes
NoWrap = Yes
```

When NoWrap=Yes, each paragraph comes out in a single line, without any line wrap. Also, leading spaces are preserved. To apply this option to a single paragraph format, see §21.3.6 [Stripping paragraph properties](#) on page 650.

13.6.5 Preventing adjacent <pre> elements from merging

By default, for HTML output **Mif2Go** merges successive elements mapped to <pre>, provided they are assigned the same CSS class. To prevent such elements from being merged:

```
[HTMLOptions]
; MergePre = Yes (default, merge adjacent pre elements, or No
MergePre = No
```

When MergePre=No, adjacent <pre> elements are not merged, even if they lack a CSS class assignment or have the same CSS class.

13.7 Defining and mapping colors for HTML

You can adjust colors in HTML output in the following ways, without using CSS:

- Map colors defined in your FrameMaker document to new colors.
- Define new colors.
- Apply any color named in your document or defined in a configuration file to:
 - paragraph and character formats (see §21.9 [Specifying text colors for HTML](#) on page 669)
 - tables (see §24.4.11 [Using shading and color in tables](#) on page 745).

In this section:

§13.7.1 [Converting colors](#) on page 438

§13.7.2 [Mapping FrameMaker colors to new values](#) on page 439

§13.7.3 [Defining new colors](#) on page 440

§13.7.4 [Using Web-safe colors](#) on page 440

§13.7.5 [Redefining colors via conversion template](#) on page 440

§13.7.6 [Understanding CMYK-to-RGB conversion anomalies](#) on page 441

13.7.1 Converting colors

Mif2Go converts the color values in your document from the CMYK model used in FrameMaker to the RGB model used in HTML. If you used colors that are not in the tiny Web-safe group (see §13.7.4 [Using Web-safe colors](#) on page 440), this conversion can result in a display that looks very different from the way it looked in FrameMaker. The difference is especially noticeable if the colors you used are from a color library, such as

one of the Pantone libraries; see §13.7.6 [Understanding CMYK-to-RGB conversion anomalies](#) on page 441.

Specify exact colors

To correct the colors, for each color named in FrameMaker you can specify the exact RGB value you want to use, in section [Colors]. You *must* use [Colors] to get the results you want from any color that originated in one of the FrameMaker color databases. However, if you can specify RGB colors to begin with instead of CMYK in your FrameMaker document, you might not need to redefine them in [Colors].

Define Web-safe colors

If your colors are already CMYK, you cannot just change them to RGB in FrameMaker; you must edit the percentages for every single color, even if they seem right. Use multiples of 20% for all values, so the colors are all in the Web-safe set. FrameMaker still writes CMYK to the MIF intermediate files, but it is CMYK for which the **Mif2Go** conversion algorithm works. See §13.7.6 [Understanding CMYK-to-RGB conversion anomalies](#) on page 441 for more information.

13.7.2 Mapping FrameMaker colors to new values

To specify color values:

```
[Colors]
; color name = hex RGB value (names as used by Frame), or
; color number 1-254 = hex RGB value; color number 0 is invisible
; 1..8 = black, white, red, green, blue, cyan, magenta, yellow
; more may be defined in the Frame file; this overrides them
; numbers up to 254 may be defined here and used in [HTML*Styles]
; for example, 100=804000 defines 100 as olive brown
```

The reserved color numbers have the following hexadecimal RGB values:

1	black	000000
2	white	FFFFFF
3	red	FF0000
4	green	008000
5	blue	0000FF
6	cyan	00FFFF
7	magenta	FF00FF
8	yellow	FFFF00

When you define a color in FrameMaker, start the color name with a letter. If the first character in the name is a digit, **Mif2Go** assumes the entry is a color number.

To map colors, assign hexadecimal RGB values to either of the following:

- Colors you defined in your FrameMaker document, by color name.
- Default FrameMaker colors, by color number; FrameMaker color numbers for default colors are listed in [Table 13-2](#).

Table 13-2 Color numbers for default FrameMaker colors

Color:	Black	White	Red	Green	Blue	Cyan	Magenta	Yellow
Number:	1	2	3	4	5	6	7	8

If you defined a color in FrameMaker and want to adjust the color for HTML, you can redefine it; for example:

```
[Colors]
; Change SeaWater to a Web-safe color
SeaWater=0099CC
```

To map a default FrameMaker color to some other color, assign its number a new RGB value; for example:

```
[Colors]
; Replace Cyan with SeaWater
6 = 0099CC
```

See also §21.9 [Specifying text colors for HTML](#) on page 669.

*Color mappings
do not affect
illustrations*

The colors you map in the [Colors] section are not applied to illustrations created with FrameMaker drawing tools. To change colors in FrameMaker illustrations, you must redefine the colors in a conversion template; see §2.4 [Importing formats from a conversion template](#) on page 67. However, see §13.7.5 [Redefining colors via conversion template](#) on page 440 for a reason not to do so.

13.7.3 Defining new colors

To define a new color that is not named in your FrameMaker document, assign a hexadecimal RGB value to any decimal integer from 9 through 254 (0 through 8 are reserved). Start with a number well above those assigned to colors in FrameMaker; 100 is a good choice. For example:

```
[Colors]
; Use deep pink for table headings
105 = FF3399
```

13.7.4 Using Web-safe colors

It is best to use Web-safe colors for HTML; otherwise you could run into browser palette issues. For example, if you do not specify a Web-safe value for Netscape, you get white. Because there are only 216 Web-safe colors, you are almost certain to have to redefine any colors you added to FrameMaker. However, the FrameMaker default colors are all Web safe.

For Web-safe colors that are rendered the same by all browsers, use color numbers 1 through 8, or define colors with elements of 00, 33, 66, 99, CC, or FF, which correspond to levels of 0%, 20%, 40%, 60%, 80%, and 100% in FrameMaker RGB settings. For example, RGB hexadecimal value 0099CC yields the color shown in [Figure 13-2](#).

Figure 13-2 RGB color 0099CC



[Table 13-3](#) lists the values you can use to define Web-safe RGB colors, in three different units of measurement.

Table 13-3 Ways to express Web-safe RGB color values

Units	Web-safe values for Red, Green, or Blue						Where used
Percent	0%	20%	40%	60%	80%	100%	FrameMaker color definitions
Hexadecimal	00	33	66	99	CC	FF	HTML; Mif2Go [Colors]
Decimal	0	51	102	153	204	255	Windows, some applications

13.7.5 Redefining colors via conversion template

If you are using a conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67), you could redefine colors there to achieve Web-safe colors. If all your color definitions start with a basic FrameMaker color, such as Black, that can work well. However, if the colors in your original document came from a color library such as

one of the Pantone libraries, this leaves you at the mercy of incorrect color translation in FrameMaker; see §13.7.6 [Understanding CMYK-to-RGB conversion anomalies](#) on page 441. **Mif2Go** cannot tell which way a given color originated, either in FrameMaker or in MIF.

13.7.6 Understanding CMYK-to-RGB conversion anomalies

CMYK and RGB are fundamentally different. CMYK is subtractive. It defines a color in terms of how the color reflects or absorbs light (most useful for printing). RGB is additive. It defines a color in terms of the light that is emitted (most useful for monitors). Any conversion from one to the other is bound to be an approximation.

However, **Mif2Go** encounters a bizarre CMYK-to-RGB conversion problem with colors defined in color libraries. HTML requires RGB. FrameMaker uses CMYK internally, though RGB is used for the screen display in Windows. The values that FrameMaker uses for CMYK do not always convert to the same RGB values as those FrameMaker displays; often, nowhere near the same.

If you choose a Pantone color, for example, and view it in RGB in FrameMaker, the percentages you see are not always the correct computed value (which is what **Mif2Go** uses). Also, you can define two visually very different colors that appear to have the same CMYK value in FrameMaker, but different RGB values.

This is different from what happens if you define a *new* color based, say, on black; you can work in either RGB or CMYK, and when you flip back and forth, FrameMaker uses the correct conversion (the same conversion **Mif2Go** uses).

But FrameMaker's color databases do something very odd. If you look at a CMYK color that has no black, such as PANTONE 528 CVC (a maroon), and note the Cyan, Magenta, and Yellow percentages; then change to RGB, and note the Red, Green, and Blue percentages; each matching pair (Cyan-Red, Magenta-Green, Yellow-Blue) should total 100%, because aside from black, the two color models are mathematical complements. However, what you see instead are the percentages shown under **FrameMaker RGB** in [Table 13-4](#). The color still looks maroon in FrameMaker.

With model **RGB** selected, choose **Black**, then choose **New Color**, and enter the *computed* values, from the **Computed RGB** column in [Table 13-4](#). Change to model **CMYK**. The percentages now match the original FrameMaker CMYK values for PANTONE 528 CVC, but the color is no longer maroon, it is a deep violet/purple—with exactly the same CMYK values as the maroon.

Table 13-4 FrameMaker color conversion anomaly

Percentages for PANTONE 528 CVC (maroon)					
FrameMaker CMYK		Computed RGB		FrameMaker RGB	
Cyan:	43.0	Red:	57.0	Red:	58.3
Magenta:	56.0	Green:	44.0	Green:	35.5
Yellow:	0.0	Blue:	100.0	Blue:	64.1

13.8 Converting generated files for HTML

You can direct **Mif2Go** to include any of contents, index, and other generated files in the output. For HTML/XHTML/XML, instead of generating these lists anew from markers and cross references, **Mif2Go** converts your FrameMaker-generated files.

In this section:

§13.8.1 [Converting FrameMaker IX and other marker lists](#) on page 442

§13.8.2 [Converting FrameMaker TOC and other paragraph lists](#) on page 444

13.8.1 Converting FrameMaker IX and other marker lists

Links in FrameMaker index files, including the standard IX file and any other IOM (index-of-markers) files, point to markers in your FrameMaker document. **Mif2Go** can convert the links to produce HTML index files.

In this section:

§13.8.1.1 [Including standard and other index files](#) on page 442

§13.8.1.2 [Including markers other than Index](#) on page 442

§13.8.1.3 [Replacing page numbers with symbols or images](#) on page 442

§13.8.1.4 [Making index entries into links](#) on page 443

§13.8.1.5 [Correcting or suppressing links in <\\$npage> index entries](#) on page 444

13.8.1.1 Including standard and other index files

To include in HTML output a standard FrameMaker IX file, check **Use FM IX** in the *Set Up* dialog (see §3.4 [Choosing project set-up options](#) on page 79), or specify the following setting:

```
[Setup]
UseFrameIX = Yes
```

Other IOM files

To include other FrameMaker-generated IOM (index-of-markers) files, also check **Use other FM generated files** in the *Set Up* dialog, or specify the following setting:

```
[Setup]
UseFrameGenFiles = Yes
```

*Apply a character
format to page
numbers*

To get active links for multiple-page entries in output from FrameMaker IX and other IOM files, you must apply a character format to the page numbers. See §5.5 [Converting FrameMaker-generated files](#) on page 124.

13.8.1.2 Including markers other than Index

If your index files reference FrameMaker markers of types other than **Index**, map those markers to Index markers (see §29.3 [Remapping marker types and hypertext commands](#) on page 836). For example:

```
[Markers]
Subject = Index
SpecialKey = Index
SectName = Index
```

13.8.1.3 Replacing page numbers with symbols or images

You cannot just remove page numbers from an index when you convert index entries to HTML, because there are often multiple page numbers for each entry. Instead, to suppress page numbers you have to replace them with something big enough to click on. Here is one way:

1. In the FrameMaker IX file, define a new character format; for example, *IXpgnum*.
2. On the IX Reference page of the IX file, find the line with paragraph format *IndexIX*, which contains the <\$pagenum> element, and apply character format *IXpgnum* to the entire line.

3. Save the index file, and generate the book. In the Body pages of the resulting index file, you will see that the page numbers are still links but the index text is not, just as before; so locked FrameMaker documents and PDF files are unaffected.
4. In the configuration file, specify the following settings for the character format you applied:

```
[HTMLCharStyles]
IXpnum= KeepLink CodeReplace
```

The `CodeReplace` property indicates that all text with character-format `IXpnum` will be replaced by HTML code assigned to `IXpnum` in the `[CharStyleCodeReplace]` section.

The `KeepLink` property retains the first hypertext link in the replaced text.

5. Specify HTML code for the character you want to use instead of a page number:

```
[CharStyleCodeReplace]
IXpnum=<b>&#182;</b>
```

The `IXpnum=¶` setting in this example gives you a bold paragraph symbol in place of the page number, but you can use any replacement you please. For example, you could specify the following setting (using XHTML conventions):

```
[CharStyleCodeReplace]
IXpnum=
```

This setting replaces the page number with an image. Because the resulting HTML must repeat this code fragment for every index item in the document, keep the image file name very short; perhaps just `p.gif`.

To apply `KeepLink` to *all* `[HTMLCharStyles]CodeReplace` character formats listed in `[CharStyleCodeReplace]`:

```
[HTMLOptions]
; KeepReplacedCharLinks = No (default) or Yes (retain hotspot href
; when CodeReplace is used for a char format, meant for IX pagenums
; and equivalent to setting KeepLink for all char-fmt CodeReplace
; settings)
KeepReplacedCharLinks=Yes
```

This way you do not need to set `KeepLink` individually for each character format.

13.8.1.4 Making index entries into links

If your index has only one destination per entry, and you are certain this will always be the case, you can make the text of the entries act as links, and hide the page numbers. Use the procedure explained in §13.8.1.3 [Replacing page numbers with symbols or images](#) on page 442, with the following exceptions:

- Specify something that is *not* visible in place of the symbol or `` tag; for example:

```
[CharStyleCodeReplace]
IXpnum=&nbsp;
```

- Make each index-entry paragraph into a hotspot:

```
[HTMLParaStyles]
Level?IX=ParaLink

[HTMLCharStyles]
IXpnum= KeepLink CodeReplace
```

This method uses `ParaLink`, a setting meant for TOCs, to specify that the entire paragraph should be the hotspot for any link it contains, regardless of any character format

applied. The “?” in `Level?IX` is a wildcard that makes the setting apply to all level numbers.

13.8.1.5 Correcting or suppressing links in <\$nopage> index entries

A FrameMaker-generated index includes, for every <\$nopage> entry, a link to the original marker location in the document. IndexRef, a FrameMaker plug-in available from Sundorne Communications, can change the links for <\$nopage> *See* and *See also* entries in FrameMaker, so the links point to the referenced index entries instead. See:

<http://www.sundorne.com/FrameMaker/IndexRef/indexref.htm>

If you do not use IndexRef, to prevent links for <\$nopage> entries from showing up in HTML output, assign the NoHref property to `Level*IX` paragraph formats. For example:

```
[HTMLParaStyles]
; NoHref suppresses <a href=...> tags in para formats
Level1IX=NoHref
Level2IX=NoHref

[HTMLCharStyles]
zIXpgnum= KeepLink CodeReplace
```

NoHref omits links for *See* and *See also* entries when you assign KeepLink to the page-number character format; KeepLink overrides NoHref for the real page numbers.

Note: Unless you either use IndexRef or assign NoHref to `Level*IX` paragraph formats, the initial text of each <\$nopage> entry becomes a link to the location of the <\$nopage> marker in your document, not to the referenced *See* or *See also* entry in the index.

13.8.2 Converting FrameMaker TOC and other paragraph lists

Links in FrameMaker-generated list files, including the standard TOC file and any other LOP (list-of-paragraphs) files, point to the ObjectIDs of paragraphs in your FrameMaker document. Mif2Go can convert the links to produce HTML list files.

In this section:

§13.8.2.1 [Including a standard TOC and other lists](#) on page 444

§13.8.2.2 [Including paragraph references](#) on page 445

§13.8.2.3 [Eliminating page numbers from generated lists](#) on page 445

13.8.2.1 Including a standard TOC and other lists

To include in HTML output a standard FrameMaker TOC file, check **Use FM TOC** in the *Set Up* dialog (see §3.4 [Choosing project set-up options](#) on page 79), or specify the following setting:

```
[Setup]
UseFrameTOC=Yes
```

To include other FrameMaker-generated such LOT and LOF, also check **Use other FM generated files** in the *Set Up* dialog, or specify the following setting:

```
[Setup]
UseFrameGenFiles=Yes
```

See §5.5 [Converting FrameMaker-generated files](#) on page 124.

13.8.2.2 Including paragraph references

Link destinations in the files referenced by FrameMaker-generated lists consist of the ObjectIDs of the referenced paragraphs (see §5.3 [Identifying files and objects](#) on page 117). These ObjectIDs must be retained in HTML output, as ``. However, if you use the following setting recommended in §19.5.3 [Including ObjectID anchors as link targets](#) on page 620, **Mif2Go** discards any ObjectIDs that appear not to be needed:

```
[HtmlOptions]
ObjectIDs=Referenced
```

When **Mif2Go** converts a standard FrameMaker TOC file, ObjectIDs are automatically retained for any paragraphs to which you assign [HTMLParaStyles] property `Split` or `Title`. For other generated lists, or if your TOC includes links to paragraphs that are not assigned these properties, you must tell **Mif2Go** that the ObjectIDs are to be retained. You do this by assigning the `Contents` property to the referenced paragraph formats.

For example, if you are converting FrameMaker LOF and LOT files, to retain ObjectIDs for figure captions and table titles you might specify the following:

```
[HTMLParaStyles]
FigCaption=Contents
TableTitle=Contents
```

The only possible drawback is that if you use the same configuration file when you direct **Mif2Go** to *generate* Contents, such as for HTML-based help, links to these paragraph formats are added to the generated Contents also. You can override this default behavior by specifying a zero Contents level for any unwanted paragraph formats; for example:

```
[HelpContentsLevels]
FigCaption=0
TableTitle=0
```

13.8.2.3 Eliminating page numbers from generated lists

To keep page numbers from appearing in the output, you can use the following technique:

1. On the TOC Reference page of your FrameMaker TOC file (or the equivalent Reference page for any other generated-list file):
 - 1.1. Define a character format (for example, *TOCpgnum*) set to all **As is**, and add it to the character catalog.
 - 1.2. Apply the character format to all `<$pagenum>` elements and to their preceding spaces or tabs.
2. Save the TOC or generated-list file.
3. Generate the book.
4. Check the resulting Body pages of the generated file. When you hold down **Ctrl+Alt** and mouse over the contents, you should see that the text of each entry is a link, but the page number is not. This might matter if you are also distributing locked FrameMaker files, or making PDF files; however, the links still work in both cases.
5. In the configuration file, specify the following setting for the character format you applied:

```
[HTMLCharStyles]
TOCpgnum=Delete
```

The resulting HTML file will not have page numbers.

13.9 Importing HTML files as insets

To include existing HTML code as an inset, by importing HTML from a source other than the FrameMaker files you are converting, use a FrameMaker **HTML Macro** marker. The content of the marker should look like this:

```
<$.\filename.htm>
```

where *filename.htm* is the name of the HTML file you want to import. Place the marker in your FrameMaker document wherever you want the imported HTML to appear in your **Mif2Go** output.

If the HTML you are importing is not a fragment, but a complete HTML file with both `<head>` and `<body>` sections, to omit all but the `<body>` part use a **Mif2Go** macro expression (see §28.6 [Using expressions in macros](#) on page 811) in the content of the **HTML Macro** marker. Code such as the following would select all text between `<body>` and `</body>` from *filename.htm*:

```
<$(($.\\filename.htm after "<body>") before "</body>")>
```

The single dot after the second \$ indicates that the file you are importing is in the current directory; if it is in some other directory, use a full path name.

Note: You must use the two-backslash form of separator: backslash (\) instead of a forward slash (/) so that the **Mif2Go** expression evaluator does not take it as a division operator; and two of them, the first to escape the second backslash in the FrameMaker marker.

13.10 Converting conditions to HTML attributes

Mif2Go can convert FrameMaker text conditions to attributes in HTML/XHTML output elements.

In this section:

§13.10.1 [Understanding how Mif2Go converts conditions](#) on page 446

§13.10.2 [Mapping FrameMaker conditions to HTML attributes](#) on page 447

§13.10.3 [Displaying condition indicators in HTML with CSS](#) on page 447

See also:

§15.12 [Converting conditions to DITA attributes](#) on page 533

13.10.1 Understanding how Mif2Go converts conditions

If a full element (either paragraph or character, block or inline) is conditional, **Mif2Go** sets an attribute for it. If you are using CSS, **Mif2Go** modifies the existing class by concatenating the condition attribute to the original class name; for example:

```
class="body linux"
```

CSS can use such attribute lists.

If the condition does not apply to all of an element, **Mif2Go** encloses the conditional part in a pair of tags, with the same attributes. By default, **Mif2Go** uses `` for HTML. However, you can specify another tag to use for this purpose:

```
[HTMLOptions]
; ConditionCharTag = tag to interpolate for conditions that affect
; only part of the enclosing element, default span for HTML.
ConditionCharTag = tagname
```

For example:

```
<span class="linux">...</span>
```

When conditions overlap each other, or overlap inline elements, **Mif2Go** creates a new tag pair for each change, to respect HTML no-overlap rules. For example:

```
<p>This paragraph contains <span class="linux">text for </span>
<i><span class="linux">Linux </span><span class="linux windows">
as well as</span><span class="windows"> text</span></i>
<span class="windows"> for Windows</span>, with overlapping conditions
and a character format overlapping both, resulting in five
<span> elements.</p>
```

In addition to text, **Mif2Go** applies conditions to `<table>` and `` elements, and to `` elements, based on the conditions in effect in FrameMaker at the point of the table or figure anchor, cross reference, or marker.

Mif2Go supports conditional table rows; row condition attributes are not applied to the paragraphs within cells. Likewise, the attributes of block tags are not applied to inline tags enclosed within the block.

Where multiple blocks make up a larger element, **Mif2Go** does not push the attributes up to the enclosing element; they remain on the enclosed block elements.

13.10.2 Mapping FrameMaker conditions to HTML attributes

To use this feature, set FrameMaker **Show/Hide** to show all the conditions for which you want content present. The conditions shown are identified in the output. Content that remains hidden in FrameMaker is not included. Multiple conditions result in multiple conditional attribute values.

To map a FrameMaker condition to an HTML element attribute:

```
[ConditionAttributes]
; Condition name = attribute for elements, usually class.
CondName = attrname="attrvalue"
```

For HTML, the attribute name is usually `class`. For example:

```
[ConditionAttributes]
HelpOnly = class="online"
```

13.10.3 Displaying condition indicators in HTML with CSS

You can use CSS to make the FrameMaker conditions transferred to HTML attributes stand out, by setting flags to display the original condition indicators such as color, underline, and strikethrough:

```
[ConditionOptions]
; UseConditionalFlagging = No (default, do not include flags)
; or Yes (set flags per conditions)
UseConditionalFlagging = Yes
; CSSFlagsFile = name of CSS file to use for flagging classes for
; HTML outputs. If not specified, related settings below ignored.
CSSFlagsFile = flags.css
; WriteFlagsFile = Yes (default, write in project directory) or
; No (do not write)
WriteFlagsFile = Yes
; ReferenceFlagsFile = Yes (default, reference after main CSS file
; in output document head) or No (do not reference).
ReferenceFlagsFile = Yes
```

When `UseConditionalFlagging=Yes`, **Mif2Go** writes `class="condition_name"` into the HTML, or prepends `condition_name` to an existing `class` attribute. This works because a `class` attribute can have multiple values, separated by spaces. When there is a CSS rule for more than one of them, the rules are additive. If a particular property has contradictory settings, such as two different colors, the later of the two in the CSS file overrides the earlier; the order in the `class` attribute does not matter.

CSS file name is required

You must specify a name for `CSSFlagsFile` if you want the CSS file written and referenced in HTML output. By default, the CSS file is placed in the project directory.

Colors are not combined

In preparing the CSS rules for condition indicators, **Mif2Go** does not try to combine colors. If multiple conditions with different color indicators are applied to the same text in **FrameMaker**, the last applicable selector in the CSS file overrides any that precede it.

13.11 Providing hover text for terms in HTML

Hover text in HTML is produced from the value of the HTML `title` attribute of a tag (usually a `` tag) that encloses the term. The attribute value may not contain a carriage return, nor the symbols `<`, `>`, `"`, or `&`.

Mif2Go provides two ways to assign hover text to terms in your document:

[Assign hover text with a marker](#)

[Assign hover text with a character format.](#)

Assign hover text with a marker

To provide hover text for a single instance of a term, in **FrameMaker** place an attribute marker within or immediately before the character format that encloses the term. The content of the marker is the text that will show on hover. See §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.

Assign hover text with a character format

To provide the same hover text for every instance of a given term, or to provide hover text for multiple terms, apply a dedicated character format (for example *Term*) to the material in question. Assign format property `GlossTitle` to the character format:

```
[HTMLCharStyles]
Term = GlossTitle
```

Hover text is captured from content

When **Mif2Go** encounters in your document character format *Term*, **Mif2Go** does the following:

1. Collects the content enclosed by the character format.
2. Removes all characters except letters and digits.
3. Uses the result as a key to locate the required hover text.

Hover text can be in your configuration file or in separate files:

[Keep hover text in a configuration file](#)

[Keep hover text in separate text files.](#)

Keep hover text in a configuration file

If the hover text you wish to provide is relatively brief, such as spelling out the names represented by acronyms used in your document, you can list the terms and their definitions in your project or document configuration file. For example:

```
[GlossTitles]
ATT = American Telephone and Telegraph
DEC = Digital Equipment Corporation
HP = Hewlett-Packard
```

Keep hover text in separate text files

If **Mif2Go** does not find the key in configuration section `[GlossTitles]`, **Mif2Go** looks for it in the following section:

```
[GlossFiles]
ATT = att
```

If found, **Mif2Go** looks for a file by the assigned name with extension `.txt` (in this example, `att.txt`) and uses the contents of that file as the hover text for the term. If you have a set of similar terms that should use the same hover text, you can use wildcards; for example:

```
[GlossFiles]
ATT* = att
```

In this example, any term that begins with AT&T (or ATT) and has the hover-text character format or marker would get hover text from file `att.txt`.

By default, **Mif2Go** looks in the project directory for hover-text files. Most likely you will want to keep the files in some other central location. To specify where to find these files:

```
[HTMLOptions]
; GlossTitlePath = path to definition files used for HTML hover text
GlossTitlePath = C:/path/to/definitions/
```

The path must use forward slashes, and must not contain spaces.

If **Mif2Go** does not find the key listed either in `[GlossTitles]` or in `[GlossFiles]`, **Mif2Go** uses the key itself as the base file name, and looks for `key.txt`.

*Precedence of
hover-text
definitions*

The precedence of methods by which **Mif2Go** finds hover text for a term is as follows:

1. A marker, if present
2. A `[GlossTitles]` setting
3. A `[GlossFiles]` setting
4. A file named by the term content.

13.12 Generating XHTML for Confluence 4.x

XHTML output that will work as input to Confluence requires a different syntax for links, and several special settings. Thanks to research by Robert Lauriston, **Mif2Go** provides a way to produce the required markup. See:

<https://confluence.atlassian.com/display/DOC/Confluence+Storage+Format>

To direct **Mif2Go** to produce XHTML for Confluence 4.x:

```
[HTMLOptions]
; Confluence = No (default, use normal linking)
; or Yes (make Confluence links)
Confluence = Yes
```

When `Confluence=Yes`, **Mif2Go** automatically sets the options listed in [Table 13-5](#). You can override these individually if necessary.

Table 13-5 Default options for Confluence 4.x XHTML

Configuration section	Setting	Value	Reference
[HTMLOptions]	ConfluenceLinks	Yes	13.12
	RemoveANames	Yes	14.6
	NoLocations	Yes	19.3.2
	NoFonts	Yes	21.7.4
	UseHash	No	14.6
	AlignAttributes	No	21.5
	UseXMLDeclaration	No	13.4.7
	UseDOCTYPE	No	13.4.1
	UseHeadAndBody	No	13.4.7
[CSS]	UseCSS	No	22.4.2
	UseSpanAsDefault	No	22.7.3

To configure Confluence links:

```
[HTMLOptions]
; ConfluenceLinks = No (default, use normal links)
; or Yes (use the link parts specified below)
ConfluenceLinks = Yes
; These are the default parts for Confluence links:
ConfluenceLinkStart = <ac:link>
ConfluenceLinkPage = <ri:page ri:content-title="
ConfluenceLinkPageEnd = ">
ConfluenceLinkText = <ac:link-body>
ConfluenceLinkTextEnd = </ac:link-body>
ConfluenceLinkEnd = </ac:link>
```

When ConfluenceLinks=Yes, the remaining ConfluenceLink* settings are in effect.

Note: The XHTML files you produce with **Mif2Go** must be imported into Confluence one at a time. As of this writing, no batch import utility is available.

13.13 Exporting content for database input

If you are generating HTML that is destined for input to a database, you might want to exclude everything except the <body> content. You can use the following setting to generate HTML without the prolog, <html> tags, <head> tags and content, or <body> tags. This leaves just the body content, in a form suited to inclusion in a database:

```
[HTMLOptions]
; BodyContentOnly = No (default) or Yes (omit prolog, root element,
; and head and body tags, leaving only body content, for DBMS use)
BodyContentOnly = Yes
```

13.14 Using framesets

To use framesets, you must create the HTML that defines the frames yourself, possibly using another HTML tool. You can make the resulting HTML code into a **Mif2Go** macro (see §28 [Working with macros](#) on page 787), as follows:

1. Give the macro a name.
2. Copy the name and HTML code into your project configuration file or into a macro library file (m2h_macro.ini, or another macro library file you have created).

3. Include an entry for the macro in the [Inserts] section.

The generated frameset file is more a starting point than a finished product. The result might look like the following, using an example from the W3C reference on framesets:

<http://www.w3.org/TR/1999/REC-html401-19991224/present/frames.html>

The following example produces a simple three-frame layout:

```
[Inserts]
Frames=<$MyFrameset>
End=</noframes></frameset>
. . .
[MyFrameset]
<frameset cols="20%, 80%">
  <frameset rows="100, 200">
    <frame name="frame1" src="contents_of_frame1.html">
    <frame name="frame2" src="contents_of_frame2.gif">
  </frameset>
  <frame name="frame3" src="contents_of_frame3.html">
</noframes>
```

If the browser is too old to display frames, or is set not to display frames, the page contents are shown instead, as the `noframes` section.

The frameset document itself must use the frameset header:

```
[HTMLOptions]
; UseFrameSet = No (default) or Yes (if included frameset tags)
UseFrameSet=Yes
; HTMLDocType, required at start of HTML documents
; for v4 frameset is: "-//W3C//DTD HTML 4.01 Frameset//EN"
; HTMLDTD, default for v4 frameset is:
; "http://www.w3.org/TR/1999/REC-html401-19991224/frameset.dtd"
```

The frames defined in the frameset get their initial content as specified in the `src` attribute of the frame element. After that, they are reloaded by making jumps for which the `target` attribute is set to the frame name, such as:

```
<p><a href="http://www.omsys.com" target="frame3">Click here.</a></p>
```

You can set the `target` attribute for a jump by applying, to the hotspot and marker, a paragraph or character format that you list in the configuration file with a frame target name. For example:

```
[Targets]
; doc format = name of frame to use for jumps from within this format
; For OmniHelp ALink and KLink jumps, targets make no sense
; and are ignored.
Top Left=frame1
```

If the format in effect at the jump is not listed, **Mif2Go** checks to see if all jumps to that file, or to that URL destination, are intended for a particular frame. For example:

```
[TargetFiles]
; filename (no ext) or URL destination = target frame to be used
; a URL destination is the last element in the URL (no extension)
procedures=frame2
```

You can also set a default target to be used by all jumps in the file that are not otherwise set; for example:

```
[HTMLOptions]
; DefaultTarget = target to use for all jumps not otherwise set
DefaultTarget=frame3
```

To have a jump to a target open another window, you can use an HTML reserved name for the target; one such name is `_blank`, which causes opening in a new browser window. Or,

you can specify the opening method in the target file, with an `onload` attribute in the `<body>` tag.

In HTML, you can force a new window with ``, or better yet with an `href` to a JavaScript function that sets `document.location`. In any case, you get a *new* new window every time.

See also §19.4 [Creating jumps to particular windows for HTML](#) on page 616.

Note: You cannot use framesets in compiled HTML Help (.chm file); you can use them in uncompiled HTML Help only, which is of questionable value. Using framesets for HTML Help makes sense only if the result will be viewed on UNIX systems.

13.15 Adding a “Made with Mif2Go” label or button

You can include in your HTML output a small JPEG image that indicates the output was created using **Mif2Go**; [Figure 13-3](#) shows what the image looks like.

Figure 13-3 Made with Mif2Go



To include this image, invoke predefined macro `<$_madewith>` (see §28.1.4 [Using predefined macros](#) on page 792). For example, to place the image at the bottom of the first page of your HTML output:

```
[Inserts]
FirstBottom=<div align="center"><$_madewith></div>
```

You can specify the following for the “Made with **Mif2Go**” graphic:

- [Name and location](#)
- [When to produce](#)
- [Image attributes](#)
- [Enclosing tags](#)
- [Link to Omni Systems](#)

Name and location

Mif2Go produces the “Made with **Mif2Go**” image as an external graphics file named `madewithm2g.jpg`, located in the project directory. You can specify a different name and location; for example:

```
[HtmlOptions]
; MadeWithImageFile = name to use for "Made with Mif2Go" .jpg graphic
; MadeWithImageFile=madewithm2g.jpg
MadeWithImageFile=./Graphics/m2glabel.jpg
```

If you specify a different location, **Mif2Go** places the graphic in that location, and also sets the `` reference in the HTML output to point to that location. Therefore you should specify a path that will be valid on the server where you put the HTML files. You would not want to use a local absolute path, for example. Probably a relative path would be best, one that works both locally and on the server where the output is to be deployed.

When to produce

You can specify whether **Mif2Go** should write the “Made with **Mif2Go**” graphic only if you use predefined macro `<$_madewith>`, always (whether or not you use the macro), or never:

```
[HtmlOptions]
; WriteMadeWithGraphic = Macro (default, write graphic if its macro
```

```

; is used), Always (even if macro is not used), or Never (even if
; macro is used).
WriteMadeWithGraphic=Macro

```

You could produce the graphic once, by invoking the macro, then replace the predefined macro with your own macro that references the graphic; then you can set the value of WriteMadeWithGraphic to Never for future runs of the same conversion.

Image attributes You can specify attributes for the “Made with **Mif2Go**” graphic; the default attributes are as follows:

```

[HtmlOptions]
; MadeWithAttributes = attributes of <img> tag to use for macro
MadeWithAttributes=border="0" alt="Made with Mif2Go" height="48"
width="78"

```

You must list the attributes all on one line. If you change the size of the graphic, try to preserve the aspect ratio. If your output will be viewed with a Netscape browser, include a title attribute as well as the alt attribute.

Enclosing tags You can specify whether the tag for the “Made with **Mif2Go**” graphic should be enclosed in <p>...</p> tags. The default is to do so; for XML output, you would want to omit these tags:

```

[HtmlOptions]
; MadeWithPara = Yes (default, put macro inside <p>...</p> tags) or No
MadeWithPara=No

```

Link to Omni Systems By default, a link to the Omni Systems Web site is included in the “Made with **Mif2Go**” graphic; you can omit this link:

```

[HtmlOptions]
; MadeWithLink = Yes (default, use link to omsys home page in macro)
; or No
MadeWithLink=No

```

13.16 Passing W3C validation tests

Mif2Go generates W3C-valid code, and with the appropriate settings is completely conformant. The on-line version of the **Mif2Go User’s Guide**, generated with **Mif2Go**, includes on every page a W3C validation button that shows the page is valid. You can download the source files to see which settings were used; see [Availability](#) on page 41.

To check the validity of your own HTML output:

<http://validator.w3.org/>

In this section:

- §13.16.1 [Understanding limitations of W3C validation](#) on page 453
- §13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454
- §13.16.3 [Eliminating <nobr> tags](#) on page 455
- §13.16.4 [Removing full-row straddles from tables](#) on page 456
- §13.16.5 [Avoiding redundant attribute assignments in tables](#) on page 456
- §13.16.6 [Eliminating duplicate ObjectIDs](#) on page 456

13.16.1 Understanding limitations of W3C validation

Mif2Go can produce many varieties of HTML, including some that are intended for use with older browsers such as Netscape Navigator 4.x. Some default settings allow such back-compatible code generation, which does not validate to HTML 4 specifications. Also, one particular HTML tag is not accepted by W3C, certain table anomalies in your

FrameMaker document can cause validation errors, and several high-ASCII characters cause validator warnings.

If you require clean validation for your output, you might have to make some adjustments to your FrameMaker document and to your configuration settings.

13.16.2 Replacing high ASCII characters for W3C validation

W3C validation tests complain if a file includes any characters with ASCII decimal values 128 through 159. Presence of these characters does not preclude validation. However, if the file contains *real* validation errors, the W3C validator reports these characters along with the actual errors. If you fix the errors, and leave the characters, the complaint becomes just a note about “non-SGML” characters.

Note: Leaving these characters in your document does *not* make the output invalid, despite the somewhat misleading way the W3C validator lists them when something *else* in the output is not valid.

For most purposes you should not need to do anything about the characters in question. However, if you want to have **Mif2Go** remap or remove the offending characters, you can set the following option:

```
[HTMLOptions]
; ValidOnly = No (default, allow normal use of chars from 128 to 160),
; or Yes (for warning-free W3C validation, remaps or removes
; those chars)
ValidOnly=Yes
```

This option affects the following characters:

- 128 through 159 (first 32 high ASCII characters), in all fonts *except* the following:
 - Symbol
 - Zapf Dingbats
 - Webdings
- 171 and 187 (the guillemets), in macros only.

Setting ValidOnly=Yes changes the output as follows:

- curly quotes become straight quotes
- en dashes become hyphens
- em dashes become a pair of hyphens
- bullets (except those produced by tags) become mid-dots
- all other characters in the range are dropped, unless you map them yourself; see §21.5 [Assigning properties to text formats](#) on page 653.

Table 13-6 shows how **Mif2Go** treats characters in this range when ValidOnly=Yes. Depending on which version of the *Mif2Go User's Guide* you are using to view the table, some characters might not be displayed.

Table 13-6 Characters replaced or removed for W3C validation

Value	Character	Name	Replacement character (if any)
128	€	euro	<i>Removed</i>
129	(none)	(none)	<i>Removed</i>
130	,	single base quote	' 039 (single quote)
131	f	florin	<i>Removed</i>
132	"	double base quote	" 034 (double quote)

Table 13-6 Characters replaced or removed for W3C validation (continued)

Value	Character	Name	Replacement character (if any)
133	...	ellipsis	<i>Removed</i>
134	†	dagger	<i>Removed</i>
135	‡	double dagger	<i>Removed</i>
136	^	circumflex	<i>Removed</i>
137	‰	per thousand	<i>Removed</i>
138	Š	S caron	<i>Removed</i>
139	<	left single guillemet	<i>Removed</i>
140	œ	OE ligature	<i>Removed</i>
141	˘	(none)	<i>Removed</i>
142	Ž	Z caron	<i>Removed</i>
143	(none)	(none)	<i>Removed</i>
144	(none)	(none)	<i>Removed</i>
145	`	left single quote	' 039 (single quote)
146	'	right single quote	' 039 (single quote)
147	“	left double quote	" 034 (double quote)
148	”	right double quote	" 034 (double quote)
149	•	bullet	• 183 (mid-dot), except in lists
150	–	en dash	– 045 (hyphen)
151	—	em dash	– 045 (hyphen) in text, -- (two hyphens) in macros
152	~	tilde	<i>Removed</i>
153	™	trademark	<i>Removed</i>
154	š	s caron	<i>Removed</i>
155	>	right single guillemet	<i>Removed</i>
156	œ	oe ligature	<i>Removed</i>
157	˘	(varies; not used)	<i>Removed</i>
158	ž	z caron	<i>Removed</i>
159	ÿ	Y diaeresis	<i>Removed</i>
...			
171	«	left double guillemet	" 034 (double quote), in macros only
187	»	right double guillemet	" 034 (double quote), in macros only

See also:

§13.4.3 [Specifying character encoding for HTML](#) on page 431

§14.3.3 [Specifying character encoding for generic XML](#) on page 460

§21.5 [Assigning properties to text formats](#) on page 653

13.16.3 Eliminating <nobr> tags

By default, **Mif2Go** generates <nobr> tags around non-breaking hyphens. However, the <nobr> tag is not included in the W3C DTD, despite the fact that all browsers support it. To eliminate <nobr> tags from the output, specify the following option:

```
[HTMLOptions]
; AllowNobr = Yes (default, use <no> tags around nonbreaking
; hyphens, supported properly by all browsers),
; or No (required for W3C validation)
AllowNobr = No
```

13.16.4 Removing full-row straddles from tables

If you straddle all the cells in a table row in FrameMaker, you end up with a hidden “empty” row; the presence of this row is detectable only on the table context menu, which shows **Unstraddle** instead of **Straddle** when you select the row, or any cell in that row. However, MIF output generated for the table *does* include the empty row; and empty rows are not allowed for W3C validation.

You must unstraddle the entire row, remove the resulting empty row, and restraddle any cells that might still need straddling.

13.16.5 Avoiding redundant attribute assignments in tables

If you use more than one method to add the same attribute to a table, you might end up with duplicate attribute assignments, which are not allowed for W3C validation. For example, suppose you specify access method `scope` for all tables (see §26.1.3.2 [Applying the scope method to all tables](#) on page 764):

```
[Tables]
AccessMethod=Scope
```

Then if you happen to include a **CellScope** marker in some table (see §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 7720, the `scope` attribute assignment appears twice in the output for that table cell.

13.16.6 Eliminating duplicate ObjectIDs

If a file in your output contains duplicate FrameMaker ObjectIDs (see §5.3 [Identifying files and objects](#) on page 117), you get a W3C validation error. FrameMaker can get away with duplicate ObjectIDs because in addition to the number, references to objects include the format name and the entire text of the paragraph in question.

You must hunt down duplicate objects in your FrameMaker document, using clues from the validator error report and source listing. Remove one of the duplicate objects, then reinsert that object to make FrameMaker assign a new ObjectID. See §5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118.

14 Converting to generic XML

XML emphasizes document structure rather than presentation. This section shows how to generate generic XML tags and how to set XML-specific options in your project configuration file. *If you are converting to DITA XML or to DocBook XML, consult one of the following sections instead:*

§15 [Converting to DITA XML](#) on page 473

§17 [Converting to DocBook XML](#) on page 557

Topics for generic XML include:

§14.1 [Understanding how Mif2Go generates XML output](#) on page 457

§14.2 [Setting up a generic XML project](#) on page 459

§14.3 [Specifying generic XML output settings](#) on page 459

§14.4 [Providing XML tags and structure](#) on page 461

§14.5 [Converting FrameMaker lists to generic XML](#) on page 466

§14.6 [Configuring links for generic XML](#) on page 467

§14.7 [Converting graphics for generic XML](#) on page 468

§14.8 [Converting index entries to generic XML](#) on page 468

Check the *W3C Extensible Markup Language (XML) 1.0 (Second Edition)* recommendation for information about XML:

<http://www.w3.org/TR/REC-xml>

14.1 Understanding how Mif2Go generates XML output

The FrameMaker document model for an unstructured document is essentially flat. You have to write “rules” (**Mif2Go** macros) to add the tag information that establishes a hierarchy.

In this section:

§14.1.1 [Accommodating HTML features in XML output](#) on page 457

§14.1.2 [Introducing structure with Mif2Go](#) on page 458

§14.1.3 [Introducing structure with XSLT](#) on page 458

§14.1.4 [Creating structure in FrameMaker](#) on page 458

§14.1.5 [Producing SGML with Mif2Go and XSLT](#) on page 458

14.1.1 Accommodating HTML features in XML output

Mif2Go uses the same code base for generic XML as for HTML. HTML output was developed first (because XML had not been invented yet), and then XHTML. Therefore, almost any HTML feature that can be used to produce valid XML does work in XML.

Table borders specified as standard pre-CSS HTML table attributes are a special case. For many of those, **Mif2Go** reverses the defaults for XML, so that the attributes you (almost) always want in HTML are *not* automatically added in XML. See §14.4.3 [Eliminating HTML attributes and tags for generic XML](#) on page 463.

14.1.2 Introducing structure with Mif2Go

You can use the **Mif2Go** macro language to construct a hierarchical file out of a flat file, with very little code. In many ways, it is easier to write **Mif2Go** macros than to write the conversion tables FrameMaker uses to do the structuring; certainly it is no more difficult.

When you use native FrameMaker XML export via **File > Save As...** to export an unstructured document to XML, you have to map the format tags to corresponding elements. When you use **Mif2Go** to generate XML, you still have to map format names to element names, but you can also specify nesting levels and attributes; and you can use macros freely to create additional structure, before and after elements and elsewhere.

If you map only format names, what you get when you use **Mif2Go** to generate XML is one root element, which you specify in the configuration file; see §14.3 [Specifying generic XML output settings](#) on page 459. The root element contains an element for every paragraph. Each paragraph element can contain elements for each character format used in the paragraph, and you can specify nesting levels for paragraph elements.

14.1.3 Introducing structure with XSLT

You could process “raw” XML output (which is always well formed) with XSLT to produce pretty much any structure you want. Introducing structure this way has two possible drawbacks:

- XSLT processors are not fast, especially on very large documents; most are written in Java, but even the C++ implementations take longer than you might find tolerable in production.
- Designing the XSLT template to add the structure could be a guru-level task.

14.1.4 Creating structure in FrameMaker

Although the generic XML **Mif2Go** generates from unstructured FrameMaker is well formed, it cannot be made to conform to an XML DTD. Unstructured documents do not store metadata, so element attributes cannot be exported.

If your purpose is to produce DITA XML or DocBook XML, you can use **Mif2Go** directly from unstructured FrameMaker, and the result will conform to the respective DTD. See:

§15 [Converting to DITA XML](#) on page 473

§17 [Converting to DocBook XML](#) on page 557

14.1.5 Producing SGML with Mif2Go and XSLT

Because XML is an SGML application, you can use **Mif2Go** to export to XML, then use XSLT to convert the XML into any form of SGML. Or, you can make the “XML” output come out directly in your preferred SGML flavor, by using macros to produce any alterations you need.

Depending on the system you use for subsequent XML processing, you might have to suppress \n line breaks in **Mif2Go** XML output; see §13.6.4 [Suppressing line breaks in HTML and XML output](#) on page 437.

14.2 Setting up a generic XML project

For the most part **Mif2Go** conversions to XML employ the same project set-up options, conversion methods, macros, and configuration settings as conversions to HTML or XHTML; see §13.2 [Setting up an HTML project](#) on page 424.

- Set-up options* If you are using the FrameMaker plug-in version of **Mif2Go**, you get the same *Set Up* dialog for XML as for HTML; see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425.
- Conversion files* The same conversion files are generated and named the same way for XML as for HTML or XHTML; see §C.2.3.2 [HTML/XML conversion files](#) on page 1022.
- Default settings* Default values for configuration settings are the same for XML as for XHTML, with the following exceptions:

<u>Section</u>	<u>Keyword</u>	<u>XML default</u>	<u>XHTML default</u>
[CSS]	ClassIsTag	Yes	No
[Graphics]	GraphScale	No	Yes
[HTMLOptions]	AlignAttributes	No	Yes
	AllowOverrides	No	Yes
	Footnotes	Inline	Jump
	NoFonts	Yes	No
	UseAnums	No	Yes (except lists)
	UseFootXrefTag	Yes	No
	UseHeadAndBody	No	Yes
	XMLRoot	doc	html
[Tables]	UseCALSMoel	Yes	No
	CellAlignAttributes	No	Yes
	CellColorAttributes	No	Yes
	TableAttributes	No	Yes

- Generated files* Converting FrameMaker-generated files is the same process for XML as for HTML; see §13.8 [Converting generated files for HTML](#) on page 441. However, you can also produce an XML-tagged index directly from FrameMaker index markers; see §14.8 [Converting index entries to generic XML](#) on page 468.
- Markers and macros* If you are converting an unstructured document, you can use markers and **Mif2Go** macros to introduce structure; see §29 [Working with FrameMaker markers](#) on page 831 and §28 [Working with macros](#) on page 787.

14.3 Specifying generic XML output settings

To add or change any of the options described in this section, edit configuration file `_m2xml.ini`, located in the project directory.

In this section:

- §14.3.2 [Changing output XML version or file extension](#) on page 460
- §14.3.3 [Specifying character encoding for generic XML](#) on page 460
- §14.3.4 [Specifying the root element and content type](#) on page 461
- §14.3.5 [Preventing arbitrary line breaks in XML text elements](#) on page 461

14.3.1 Creating a generic XML project

To create a generic XML project:

1. Create a directory for output files, separate from the directory where your FrameMaker document is located.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens.
3. Name your project, and browse to the project directory you created in [Step 1](#) (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
4. Choose the following output type:
Generic XML
5. Check options in the *Set Up HTML/XML Project* dialog (see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425).
6. Use a text editor to edit the resulting `m2xml.ini` configuration file (see §4.1 [Working with Mif2Go configuration files](#) on page 91).

14.3.2 Changing output XML version or file extension

XML version To change the version of generic XML:

```
[HTMLOptions]
; XMLVersion default is "1.0".
XMLVersion = 1.0
```

File extension To change the output file extension:

```
[Setup]
FileSuffix = .ext
```

The default output file extension for XML files is `.xml`.

14.3.3 Specifying character encoding for generic XML

Character encoding determines the method used to represent character value greater than 0x7F (decimal 127). Such double-byte characters constitute the “high ASCII” set; whereas FrameMaker characters, except in the Japanese version, are all single-byte. The default for XML output is UTF-8:

```
[HTMLOptions]
; Encoding = UTF-8 (XML default), ISO-8859-1 (HTML default, numeric
; refs), or None (write 0x80-0xFF as single characters)
Encoding=UTF-8
; XMLEncoding default is "UTF-8", entities are used for ANSI chars
XMLEncoding=UTF-8
; NumericCharRefs = Yes (default, always use &#nnn;)
; or No (use UTF-8 for XML)
NumericCharRefs=No
```

*Entity references
for browsers*

If your XML output is to be rendered by Web browsers, be aware that even though UTF-8 is the XML standard encoding, many browsers do not support it. The **Mif2Go** default is to claim UTF-8 as the encoding, but to use numeric references of the form `&#nnn;` for all characters that would have to be encoded; this satisfies all browsers. That is, with default settings, **Mif2Go** does not actually produce any characters with values greater than 127 using the UTF-8 encoding; instead, **Mif2Go** uses entities for such characters, readable under any eight-bit encoding scheme.

The setting for `XMLEncoding` controls the *content* of the encoding attribute of the XML declaration. If you set `Encoding=UTF-8`, you get real UTF-8 encoding (two characters)

in place of the numeric character references. However, you can still force use of numeric references by also setting `NumericCharRefs=Yes`.

While `Encoding=None` is not strictly compliant, this setting can be useful in places like Russia, where almost the entire text would otherwise consist of numeric character references. `Encoding=None` provides a 6:1 reduction in such references.

See also:

§13.3 [Including starting code and entity references](#) on page 429

§13.4.3 [Specifying character encoding for HTML](#) on page 431

§13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454

§21.5 [Assigning properties to text formats](#) on page 653

14.3.4 Specifying the root element and content type

The default value for root is `doc` for generic XML. Because XML does not have `<head>` and `<body>` sections, the default is to omit these:

```
[HTMLOptions]
XMLRoot=doc
UseHeadAndBody=No
; ContentType = text/html (default for HTML and XHTML)
; or application/xml (default for XML); try not to use text/xml
; (for interoperability)
ContentType=application/xml
```

Content-Type is part of MIME, and is used by document-processing tools. Unless you know exactly what you want and need only a mechanism to specify it, leave this setting alone. For more information, see:

http://www.w3.org/Protocols/rfc1341/4_Content-Type.html

14.3.5 Preventing arbitrary line breaks in XML text elements

If you are generating XML to be imported into a system that treats `\n` line breaks as though they were paragraph breaks, you might have to prevent **Mif2Go** from introducing line breaks into XML paragraph text.

To suppress `\n` line breaks in all paragraphs:

```
[HTMLOptions]
; NoWrap = No (default, \n where space occurs) or Yes
NoWrap=Yes
```

See §13.6.4 [Suppressing line breaks in HTML and XML output](#) on page 437.

14.4 Providing XML tags and structure

In this section:

§14.4.1 [Generating XML from an unstructured document](#) on page 462

§14.4.2 [Deriving XML tags from format and class names](#) on page 462

§14.4.3 [Eliminating HTML attributes and tags for generic XML](#) on page 463

§14.4.4 [Including or excluding FrameMaker autonumbers](#) on page 465

§14.4.5 [Configuring forced returns for XML](#) on page 465

See also:

§28.10 [Using macros to fine-tune HTML or XML output](#) on page 828

14.4.1 Generating XML from an unstructured document

To use **Mif2Go** to organize paragraph elements into higher-level sections, you can specify XML tags for the start and end of each section. You can do this several ways; your choice depends on the existing structure of your FrameMaker document and the structure you want in XML. You can insert code with any or all of the following:

[Code-before and code-after macros](#)

[FrameMaker markers](#)

[Special paragraph format](#)

For the code-before/code-after method, you might have to rename paragraph formats that start and end a section, to make them unique in their usage, much like renaming *Heading1* to *Heading1 First*, *Heading1 Top of Page*, and so forth. The other two methods avoid such renaming, but require an edit in the FrameMaker document at each point where such additions are needed.

*Code-before and
code-after macros*

Provide code in a [ParaStyleCodeBefore] macro for a particular paragraph format that always starts a section, and a [ParaStyleCodeAfter] macro for the paragraph format that ends the section; see:

[§28.9.3 Surrounding or replacing text with code or macros](#) on page 822.

*FrameMaker
markers*

Insert markers that contain either code or a macro reference; see:

[§29.7 Inserting code or text with markers](#) on page 842

[§28.9.7 Using HTML Macro markers to invoke macros](#) on page 828.

*Special
paragraph format*

Dedicate a paragraph format to XML code, and use it to insert the code directly into your FrameMaker document, most likely with a condition applied; see:

[§28.9.3 Surrounding or replacing text with code or macros](#) on page 822.

A brief example

Suppose you have run-in heading format *RuninHead* that is always followed by paragraph format *RuninBody*, and you want all such instances to come out like this in XML:

```
<labeledinfo>
  <label>Content of label...</label>
  <info>Content of info...</info>
</labeledinfo>
```

You could specify the following settings:

```
[ParaTags]
RuninHead=label
RuninBody=info

[HTMLParaStyles]
RuninHead=CodeBefore
RuninBody=CodeAfter

[ParaStyleCodeBefore]
RuninHead=<labeledinfo>

[ParaStyleCodeAfter]
RuninBody=</labeledinfo>
```

14.4.2 Deriving XML tags from format and class names

To aid in mapping formats to elements for XML output from an unstructured document, by default **Mif2Go** uses the following for XML tags:

- all CSS class names
- names of any formats to which you have *not* assigned a CSS class name:
 - in CSS

	<ul style="list-style-type: none"> – in the [ParaClasses] or [CharClasses] section – in any other configuration-file section.
<i>Only catalogued formats</i>	<p>For format mapping to work, all character and paragraph format names must be present in the FrameMaker catalog for the file you are converting, including names of any character formats that are used only in FrameMaker marker text.</p> <p>To produce valid XML, Mif2Go converts all tags to valid CSS names, without spaces, non-alphanumeric characters, leading digits, or accented characters (which become unaccented).</p>
<i>Paragraph and character tags and attributes</i>	<p>Mif2Go uses any tags and attributes you assign in configuration sections [ParaTags] and [CharTags]; see §21.3.1 Assigning HTML tags and attributes to paragraph formats on page 646. To apply an attribute to an individual paragraph or character span, insert an attribute marker in the instance; see §29.2.3 Understanding attribute markers on page 834.</p> <p>You can specify which names to use for XML tags in any or all of the following ways:</p> <ul style="list-style-type: none"> Map class names to XML tags Map format names to classes Map graphic anchor format to a class.
<i>Map class names to XML tags</i>	<p>To map all CSS class names to XML tags (the default for XML output):</p> <pre>[CSS] ; ClassIsTag = No (default for HTML/XHTML) ; or Yes (default for Generic XML) ClassIsTag=Yes</pre> <p>When ClassIsTag=Yes, any class names you assigned to formats in the [ParaTags] and [CharTags] sections become XML tags; see §22.5 Understanding how CSS affects other options on page 687. If ClassIsTag=Yes, also specify [CSS]WriteClassAttributes=No; see §22.4.2 Specifying CSS options in a Mif2Go configuration file on page 684.</p>
<i>Map format names to classes</i>	<p>To explicitly map individual format names to class names:</p> <pre>[ParaClasses] or [CharClasses] ; Format name = class to use (default is based on name) ; For XML, the class is used as the tag name by default.</pre>
<i>Map graphic anchor format to a class</i>	<p>If your document uses a special paragraph format to anchor graphics, you can specify a class name for the format with the following setting:</p> <pre>[Graphics] ; GraphClass = class name to use for paras created to hold tags GraphClass=graphic</pre>
<i>Specify all margins in CSS</i>	<p>The following setting causes CSS entries to explicitly include all four margin values, even if some are zero:</p> <pre>[CSS] ; ZeroCSSMargins = No (default) ; or Yes (specify CSS margins even if zero) ZeroCSSMargins=Yes</pre>

14.4.3 Eliminating HTML attributes and tags for generic XML

You can use configuration settings to eliminate the following HTML tags and attributes:

- [Paragraph tags](#)
- [Character tags in markers](#)
- [Table attributes](#)
- [Graphics tags and attributes](#)

- Paragraph tags* Use either of the following methods to make HTML `<p> . . . </p>` tags go away:
- Best: supply your own XML tags in [ParaTags]; see §21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646
 - Use the NoPara property in [HTMLParaStyles]; see §21.3.6 [Stripping paragraph properties](#) on page 650.

Character tags in markers If you are converting index markers or other FrameMaker markers that contain character formatting, you can use a **Mif2Go** macro to skip the character formats; see §14.8.2.5 [Stripping character formats from index entries](#) on page 472.

Table attributes By default, when you specify XML as the output type, **Mif2Go** refrains from automatically generating HTML table and cell attributes (see §24.4.1 [Specifying attributes for all tables](#) on page 736), while preserving any attributes you add specifically for XML in the configuration file or in markers:

```
[Tables]
; TableAttributes = Yes (HTML default, to allow automatically
; generated border, cellspacing, cellpadding, or No (XML default,
; to exclude those while keeping any attributes explicitly added
; in the .ini or in markers)
TableAttributes=No
; CellAlignAttributes = Yes (HTML default) or No (XML default, to
; eliminate automatically generated align and valign)
CellAlignAttributes=No
; CellColorAttributes = Yes (HTML default) or No (XML default, to
; eliminate automatically generated bgcolor)
CellColorAttributes=No
```

See §24.4.7 [Eliminating automatically generated attributes](#) on page 739.

By default, **Mif2Go** places ` ` in each table cell that would otherwise be empty (either because the cell contained only an empty paragraph in FrameMaker, or because another configuration setting eliminated the content). This is because some browsers do not render correctly the borders, margins, and padding of a completely empty cell. To suppress this feature:

```
[Tables]
EmptyTbCellContent=
```

The empty value eliminates the ` `. Or, you can specify any other content; see §24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744. However, a better approach would be to define common entities such as ` ` in your XML DTD; see §13.3 [Including starting code and entity references](#) on page 429.

Graphics tags and attributes To eliminate width and height attributes from images:

```
[Graphics]
; GraphScale = Yes to put out width and height attributes,
; or No to eliminate them all (default for Generic XML)
GraphScale = No
```

If you do not specify any setting for GraphScale, you get the correct default for either HTML or XML.

To eliminate paragraph tags around graphics:

```
[Graphics]
; GraphWrapPara = Yes (default, wrap graphics that are not inline in
; paragraph tags) or No (eliminate wrapping tags)
GraphWrapPara = No
```

14.4.4 Including or excluding FrameMaker autonumbers

By default, **Mif2Go** omits all autonumbers from XML output:

```
[HTMLOptions]
; UseAnums = Yes (HTML default, use unless list type)
; or No (XML default)
UseAnums = No
```

To include autonumbers in XML output for selected paragraph formats:

```
[HTMLParaStyles]
; Anum includes Frame autonumber, default omits it
ParaFmt = Anum
```

To exclude autonumbers from XML output only for selected paragraph formats:

```
[HTMLOptions]
UseAnums = Yes

[HTMLParaStyles]
; NoAnum excludes autonumber in non-list items, default keeps it
ParaFmt = NoAnum
```

See also:

§14.5 [Converting FrameMaker lists to generic XML](#) on page 466

§21.5 [Assigning properties to text formats](#) on page 653.

14.4.5 Configuring forced returns for XML

By default, for XML output **Mif2Go** converts each forced return (FrameMaker **Shift+Enter**, a typesetting “hard return”) to a space, except for text within preformatted tags. Within preformatted tags, a forced return always becomes a line break (not a paragraph break).

To always convert forced returns into `
`s instead of spaces for XML (and XHTML):

```
[HTMLOptions]
; UseXMLbr = No (default) or Yes (use <br /> in XML and XHTML outputs
; for hard returns)
UseXMLbr = Yes
```

Or, you can tell **Mif2Go** to close the paragraph tag at a forced return instead of substituting a space, and then reopen the tag without attributes:

```
[HTMLOptions]
; XMLBreakPara = No (default, each Shift+Enter becomes a space)
; or Yes (close para tag and reopen without attributes).
XMLBreakPara = Yes
```

`XMLBreakPara` applies only to text destined for non-preformatted tags in XML output. This setting does not affect XHTML or HTML output.

When `XMLBreakPara=No`, **Mif2Go** changes each forced return (FrameMaker **Shift+Enter**) within a non-preformatted paragraph to a space.

When `XMLBreakPara=Yes`, **Mif2Go** closes the paragraph tag, and then reopens the tag without attributes.

To override the setting for `XMLBreakPara` for selected paragraph formats:

```
[HTMLParaStyles]
; These two apply to XML, but not to XHTML or HTML:
; XMLBreak closes the current para tag at each Shift+Enter, and
; reopens it without attributes; overrides
; [HTMLOptions]XMLBreakPara=No.
```

```

; XMLNoBreak replaces each Shift+Enter with a space; overrides
; [HTMLOptions]XMLBreakPara=Yes.
ParaFmt = XMLBreak

```

For XHTML or HTML output, see §21.3.8 [Deciding how to treat forced returns](#) on page 651.

14.5 Converting FrameMaker lists to generic XML

Just converting an unstructured FrameMaker list to XML provides nothing except XML tags around each list item. Autonumbers are converted to text. Bullets become entity references. List properties you would supply for HTML have no meaning in XML.

Suppose you want to convert FrameMaker autonumbered lists that use paragraph formats such as *StepFirst* and *StepNext*; and suppose you want the lists to be structured like this in XML:

```

<steplist>
  <stepitem>
    <steptext value="1">Text of first step.</steptext>
  </stepitem>
  <stepitem>
    <steptext value="2">Text of second step.</steptext>
  </stepitem>
  . . .
</steplist>

```

You can remove the FrameMaker autonumbers by assigning property `NoAnum` to the step formats, and instead use a macro variable (`$$StepNum` in the following examples) as a counter to generate step numbers. Assign `CodeBefore` and `CodeAfter` properties to both formats, so you can surround the text with XML tags. Also assign `NoPara` to suppress any HTML `<p>` tags:

```

[HTMLParaStyles]
; Remove the autogenerated numbers:
StepFirst=CodeBefore CodeAfter NoAnum NoPara
StepNext=CodeBefore CodeAfter NoAnum NoPara
; Make sure all non-step formats can close the list:
*=CodeBefore

```

Assign code to both formats to surround the text with XML tags. If you are not using a list-ending format in FrameMaker (such as *StepLast*), you must provide a list-closing tag in code that immediately precedes any non-step format. Therefore, *all non-step* paragraph formats must have the `CodeBefore` property; thus the final wildcard setting in `[HTMLParaStyles]`.

Provide code to start the list, start each list item, and start the list-item counter:

```

[ParaStyleCodeBefore]
; Start the list:
StepFirst=<steps>\n<stepitem><$$StepNum = 1><steptext value="1">
; Start each list item:
StepNext=<stepitem><steptext value="<$$StepNum++ as %s">
; End the list before the first non-list paragraph:
*=<$_if ($$StepNum)></steps>\n<$$StepNum = 0><$_endif>

```

Whenever a non-step paragraph is encountered, if a list was in progress, the last setting in `[ParaStyleCodeBefore]` provides closing XML tags for it, and resets the step counter.

If you have more than one paragraph in any step, you have more work to do; you must provide explicit “before” and “after” code for each paragraph format that appears within a

list, to prevent those formats from terminating the list. Or, you could use a *StepLast* format for the final item in each list, and avoid this problem.

The “after” code for each paragraph format in the list provides the closing XML tags:

```
[ParaStyleCodeAfter]
; End each list item:
Step*=</steptext></stepitem>
```

If you use a *StepLast* format, you could end the list with it here, instead of using the wildcard settings in [ParaStyleCodeBefore]. You would place the following setting *ahead of* the Step* setting in [ParaStyleCodeAfter]:

```
StepLast=</steptext></stepitem></steplist>\n<$$StepNum = 0>
```

To account for the possibility that a list item is the very last paragraph in a document, check for a list at the very end:

```
[Inserts]
; This handles the case where a list ends the document:
End = <$_if ($$StepNum)></steplist>\n<$$StepNum = 0><$_endif>
```

Again, if you consistently use a *StepLast* format, you can omit the [Inserts]End setting.

Because the wildcard setting in [ParaStyleCodeBefore] most probably tests the value of \$\$StepNum before any other code has a chance to assign a value to this variable, you must explicitly give it a starting value:

```
[MacroVariables]
; Because you test before you set, you must initialize $$StepNum:
StepNum = 0
```

Again, if you use a *StepLast* format, you can omit this setting.

14.6 Configuring links for generic XML

There is no standard way to represent links in XML. Configure links whatever way your DTD or schema says; anything “well formed” is valid. See *W3C XML Pointer, XML Base and XML Linking* for more information:

<http://www.w3.org/XML/Linking>

To configure links for DocBook XML, see §17.3.3 [Configuring links for DocBook XML](#) on page 563. To configure links for DITA XML, see §15.10 [Configuring cross references and links for DITA](#) on page 527.

To manage links and cross references in generic XML:

```
[HTMLOptions]
; These are mainly intended for making links for Generic XML use:
; RemoveANames = No (default) or Yes (eliminate <a name=...> tags)
; RemoveATags = No (default) or Yes (eliminate <a href=...> tags)
; RemoveAHrefAttrs = No (default)
; or Yes (remove href attrs, keep tags)
; XMLLinkAttrs = No (default)
; or Yes to add attrs to <a href=...> tags:
; xml:link="simple" show="replace" actuate="user" class="url"
XMLLinkAttrs=No
; ATagElement = tag to use for all link elements, default is "a"
; except for DITA, where it is "xref"
ATagElement=a
; HrefAttribute = name to use for link source attr, default href
HrefAttribute=href
; UseHash = Yes (default, start local hrefs with #) or No
UseHash=Yes
```

```

; UseUlink = No (default, use ATagName for URLs) or Yes (use
; ulink for URLs, and url as the HrefAttribute within them)
UseUlink=No
; RemoveXrefHotspots = No (default) or Yes (remove hotspot text for
; xrefs and hyperlinks to Frame files, retain it for external URLs)
RemoveXrefHotspots=No
; UseListedXrefFilesOnly = No (default) or Yes (consider any xref
; target files not listed in [XrefFiles] to refer to the current
; file.) This suppresses filenames for DocBook where files are in the
; same DocBook book; files not in the book must be listed in
; [XrefFiles].
UseListedXrefFilesOnly=No

```

Page numbers **Mif2Go** retains page numbers from your FrameMaker document as targets for hypertext **gotopage** jumps.

See also:

§15.10 [Configuring cross references and links for DITA](#) on page 527

§17.3.3 [Configuring links for DocBook XML](#) on page 563

§19.2.6 [Forcing link text to lowercase](#) on page 613

14.7 Converting graphics for generic XML

In all important respects, graphics output for XML is the same as for XHTML or HTML; see §23 [Including graphics in HTML](#) on page 703. However, when you use **Save As XML** in FrameMaker, the image references produced look like the following:

```
<IMAGE xml:link="simple" href="some.gif" show="embed" actuate="auto"/>
```

To reproduce this effect when you convert to XML with **Mif2Go**, or to provide an equivalent, you can specify values for the following graphics options:

```

[Graphics]
;ImgTagElement = tag to use for all image elements, default is "img"
;ImgTagElement=img
;ImgSrcAttr = name to use for all image source attrs, default is "src"
;ImgSrcAttr=src
; XMLGraphAttrs = No (default)
; or Yes to add attrs to XML <img .../> tags:
; xml:link="simple" show="embed" actuate="auto"
XMLGraphAttrs=No

```

Whatever values you specify apply to all graphics in your document.

To eliminate graphics from XML output, see §23.4.5 [Omitting graphics from HTML or XML output](#) on page 708.

14.8 Converting index entries to generic XML

Suppose you want to convert (non-structured) FrameMaker index markers to XML, and suppose you want the XML rendered as follows:

- a different element for each index-entry level: <index1>, <index2>, and so forth
- a parent element for the whole entry: <indexterm>.

You can do this with a combination of configuration settings and **Mif2Go** macros.

In this section:

§14.8.1 [Configuring index markers for conversion to XML](#) on page 469

§14.8.2 [Defining macros to process index content](#) on page 469

14.8.1 Configuring index markers for conversion to XML

To convert FrameMaker index markers, you need ways to extract the content of each marker and embed the text in appropriate XML elements. Because you cannot redefine the behavior of FrameMaker markers of type **Index**, you must first clone these markers. Then you can do the following:

- Surround the cloned content with **Mif2Go** macro code.
- Assign the cloned content to a **Mif2Go** macro variable for further parsing.

*New markers for
XML index*

Clone index markers, creating a new marker type to which you can assign properties; see §29.3 [Remapping marker types and hypertext commands](#) on page 836:

```
[Markers]
; Create a new marker type, cloning existing markers of type Index:
Index = NewIndex
```

*Embed new
markers in code*

Assign the Code property to marker type **NewIndex**; see §29.4 [Defining and redefining marker behavior](#) on page 838:

```
[MarkerTypes]
; NewIndex marker content is to be surrounded by macro code:
NewIndex = Code
```

Note: If **Index** marker content includes non-alphanumeric characters, to ensure proper encoding you must assign marker type property **Text** instead of **Code**; see §29.7.3 [Processing marker content as text for XML/HTML/XHTML](#) on page 844.

*User variable for
marker content*

Set the initial value of user variable `$$IXtext` to the content of each **NewIndex** marker; see §29.7.2 [Surrounding marker content with code](#) on page 843:

```
[MarkerTypeCodeBefore]
; User variable $$IXtext gets the original index-marker content:
NewIndex= <$$IXtext = "
```

*XML tags for
parent element*

Provide surrounding tags for the resulting XML element `<indexterm>`:

```
[MarkerTypeCodeAfter]
; The result of processing <$$IXtext> becomes XML element <indexterm>:
NewIndex= "><indexterm><$ProcIXtext></indexterm>
```

Macro `[ProcIXtext]` parses the content of each index entry to produce XML `<indexN>` elements. Several alternate definitions of `[ProcIXtext]` are described in §14.8.2 [Defining macros to process index content](#) on page 469.

14.8.2 Defining macros to process index content

You will need to parse each index entry to divide it into levels, because the content of each level becomes a separate XML `<indexN>` element. **Mif2Go** macro `[ProcIXtext]` handles this chore. Those parts of the macro that actually end up in output are highlighted in color: **green** for element content, **magenta** for element tags. Lines are numbered for reference.

In this section:

§14.8.2.1 [Configuring macro definitions for easier reading](#) on page 470

§14.8.2.2 [Generating XML tags for each level](#) on page 470

§14.8.2.3 [Detecting colons used as text or punctuation](#) on page 470

§14.8.2.4 [Using an alternate macro to generate XML tags](#) on page 471

§14.8.2.5 [Stripping character formats from index entries](#) on page 472

14.8.2.1 Configuring macro definitions for easier reading

*Omit spaces and
line breaks*

You might want to configure macro definitions so they can be indented like program code for easier reading; see §28.1.1.4 [Managing line breaks in macro definitions](#) on page 789:

```
[Macros]
; Ignore macro linebreaks and subsequent leading whitespace,
; to allow indenting complex macro definitions:
OmitMacroReturns=Yes
```

Note: This setting is crucial if you want to use leading spaces in **Mif2Go** macro definitions, as in the examples in §14.8.2 [Defining macros to process index content](#) on page 469.

14.8.2.2 Generating XML tags for each level

Check each index entry for colons: level separators that indicate a new index level. When a colon is encountered, close the XML element for the current level, and increment the level number:

	[ProcIXtext]	
1	<\$\$IXlevel = 1>	
2	<\$_while (\$\$IXtext length)>	
3	<index<\$\$IXlevel>>	Opening XML tag
4	<\$\$IXpart = (\$\$IXtext before ":")>	
5	<\$\$IXpart>	Text
6	<\$\$IXpart = (\$\$IXtext after ":")>	
7	<\$\$IXtext = \$\$IXpart>	
8	</index<\$\$IXlevel>>	Closing XML tag
9	<\$\$IXlevel++>	
10	<\$_endwhile>	

The unary length operator in [line 2](#) gets the length of the preceding value of variable `$$IXtext`, so the `<$_while>` expression works correctly; see:

§28.6.4 [Using control structures in expressions](#) on page 815

§28.6.5 [Specifying substrings in expressions](#) on page 817.

Assignment statements by default do not produce output, so only the value of the `$$IXpart` variable on [line 5](#) actually ends up as content between the XML `<indexn>` tags, which are output on [line 3](#) and [line 8](#); see:

§28.3.2 [Assigning values to macro variables](#) on page 797

§28.6.3 [Displaying expression results in output](#) on page 813.

14.8.2.3 Detecting colons used as text or punctuation

Suppose some FrameMaker index entries include colons as punctuation or as part of the text, instead of as level separators; that is, some colons are escaped thus `\:` in index markers. You could modify macro `[ProcIXtext]` as follows, to distinguish between colon-as-text and colon-as-separator:

	[ProcIXtext]	
1	<\$\$IXlevel = 1>	
2	<\$_while (\$\$IXtext)>	
3	<index<\$\$IXlevel>>	Opening XML tag
4	<\$\$IXpart = (\$\$IXtext before ":")>	
5	<\$_if ((\$\$IXpart last 2) is "\\")>	Check for colon-as-text
6	<\$ProcIXLitColon>	Call another macro
7	<\$_endif>	
8	<\$\$IXpart>	Text output
9	<\$\$IXpart = (\$\$IXtext after ":")>	
10	<\$\$IXtext = \$\$IXpart>	

11	</index<\$\$IXlevel>>	Closing XML tag
12	<\$\$IXlevel++>	
13	<\$_endwhile>	

You need a double backslash `\\` on [line 5](#), because that is how an escaped colon in a FrameMaker marker is represented internally by **Mif2Go** DCL.

Index-entry text that does *not* include a colon-as-text is output on [line 8](#), and surrounding XML tags are output on [line 3](#) and [line 11](#).

Process colon as punctuation

A second macro [ProcIXLitColon] has to be invoked on [line 6](#), because you cannot nest a `<$_while>` inside another `<$_while>` in the same macro; see §28.6.4.3 [Using loop structures](#) on page 816:

	[ProcIXLitColon]	
1	<\$_while ((\$\$IXpart last 2) is "\\")>	
2	<\$(\$\$IXpart first ((\$\$IXpart length) - 2)) as %s:>	Text output
3	<\$\$IXpart = (\$\$IXtext after ":")>	
4	<\$\$IXtext = \$\$IXpart>	
5	<\$\$IXpart = (\$\$IXtext before ":")>	
6	<\$_endwhile>	

Index-entry content at the current level, up to the colon-as-text, and then the text colon itself, are output on [line 2](#) of [ProcIXLitColon].

14.8.2.4 Using an alternate macro to generate XML tags

As an alternative, a somewhat simpler version of macro [ProcIXtext] uses the trim operator instead of the length operator, and steps through the index entry character by character; see §28.6.5 [Specifying substrings in expressions](#) on page 817:

	[ProcIXtext]	
1	<\$\$IXlevel = 1>	
2	<\$_while (\$\$IXtext)>	
3	<index<\$\$IXlevel>>	Opening XML tag
4	<\$_repeat>	
5	<\$_if (\$\$IXtext is "")><\$_break>	
6	<\$_elseif ((\$\$IXtext first 3) is "\\:")>	Check for colon-as-text
7	:	Text colon output
8	<\$\$IXtext = (\$\$IXtext trim first 3)>	
9	<\$_elseif ((\$\$IXtext first 1) is ":")>	
10	<\$\$IXtext = (\$\$IXtext trim first 1)>	
11	<\$_break>	
12	<\$_else>	
13	<\$(\$\$IXtext first 1)>	Text character output
14	<\$\$IXtext = (\$\$IXtext trim first 1)>	
15	<\$_endif>	
16	<\$_endrepeat>	
17	</index<\$\$IXlevel>>	Closing XML tag
18	<\$\$IXlevel++>	
19	<\$_endwhile>	

Only one macro required

This version uses `<$_repeat>` for the inner loop, which allows the inner loop to be nested in the same macro, eliminating the need for a second macro; see §28.6.4.3 [Using loop structures](#) on page 816.

Colons used as text are output on [line 7](#); all other text is output on [line 13](#), one character at a time. Surrounding XML tags are output on [line 3](#) and [line 17](#).

14.8.2.5 Stripping character formats from index entries

Still another version of macro [ProcIXtext] removes character-format names from index-marker content, as well as processing colons:

	[ProcIXtext]	
1	<\$\$IXlevel = 1>	
2	<\$_while (\$\$IXtext)>	
3	<index<\$\$IXlevel>>	Opening XML tag
4	<\$_repeat>	
5	<\$_if not (\$\$IXtext)><\$_break>	
6	<\$_elseif (\$\$IXtext starts "\\:")>	
7	:	Text colon output
8	<\$\$IXtext = (\$\$IXtext trim first 3)>	
9	<\$_elseif (\$\$IXtext starts ":")>	
10	<\$\$IXtext = (\$\$IXtext trim first)>	
11	<\$_break>	
12	<\$_elseif (\$\$IXtext starts "<")>	
13	<\$\$IXtext = (\$\$IXtext after ">")>	
14	<\$_else>	
15	<\$(\$\$IXtext first)>	Text character output
16	<\$\$IXtext = (\$\$IXtext trim first)>	
17	<\$_endif>	
18	<\$_endrepeat>	
19	</index<\$\$IXlevel>>	Closing XML tag
20	<\$\$IXlevel++>	
21	<\$_endwhile>	

This version uses a negative conditional expression to check for the end of each index entry, and uses the `starts` operator instead of the `first` operator; see:

§28.6.4.2 [Using conditional expressions](#) on page 815

§28.6.5 [Specifying substrings in expressions](#) on page 817)

Text colons are output on [line 7](#); all other text is output on [line 15](#), one character at a time.

Some `first` ([line 15](#)) and `trim first` ([line 10](#) and [line 16](#)) expressions in this version appear without a second operand; these expressions assume an implied value of 1 (one) for the second operand.

15 Converting to DITA XML

Mif2Go generates topics and maps for DITA (Darwin Information Typing Architecture) XML output, converted from either structured or unstructured FrameMaker documents. This section shows how to configure DITA-specific options. Topics include:

- §15.1 [Generating DITA XML with Mif2Go](#) on page 473
- §15.2 [Setting up a DITA XML project](#) on page 478
- §15.3 [Specifying general options for DITA](#) on page 483
- §15.4 [Configuring DITA elements](#) on page 486
- §15.5 [Nesting DITA block elements](#) on page 501
- §15.4 [Configuring DITA elements](#) on page 486
- §15.5 [Nesting DITA block elements](#) on page 501
- §15.6 [Converting tables to DITA XML](#) on page 510
- §15.7 [Specifying options for images in DITA XML](#) on page 516
- §15.8 [Organizing DITA topics](#) on page 519
- §15.9 [Configuring DITA topics](#) on page 522
- §15.10 [Configuring cross references and links for DITA](#) on page 527
- §15.11 [Exporting FrameMaker variables to DITA XML](#) on page 530
- §15.12 [Converting conditions to DITA attributes](#) on page 533
- §15.13 [Marking FrameMaker text insets in DITA](#) on page 534
- §15.14 [Including CSH targets in DITA XML](#) on page 535
- §15.15 [Overriding DITA settings with markers](#) on page 536

See also:

- §16 [Configuring DITA maps](#) on page 539
- §32 [Working with content models](#) on page 905

15.1 Generating DITA XML with Mif2Go

Before you set up a **Mif2Go** DITA project, be clear about what level of familiarity with DITA you need, what you intend to do with the output, and what role you want **Mif2Go** to play in producing DITA output.

In this section:

- §15.1.1 [Understanding the complexity of a DITA conversion project](#) on page 473
- §15.1.2 [Understanding what you need to know about DITA](#) on page 474
- §15.1.3 [Clarifying your purpose for creating DITA output](#) on page 474
- §15.1.4 [Converting from structured vs. unstructured FrameMaker](#) on page 475
- §15.1.5 [Understanding what information you must supply](#) on page 476
- §15.1.6 [Understanding how Mif2Go generates DITA output](#) on page 476
- §15.1.7 [Creating valid DITA XML output](#) on page 477

15.1.1 Understanding the complexity of a DITA conversion project

Be aware that conversion to another source format, such as DITA XML, can be difficult, especially if you are converting an unstructured document. There are no shortcuts. You might need days or weeks to get it right, working with small test documents, before you can go into production.

You can view a Scriptorium Publishing webinar about the process, showing several alternatives:

<http://bit.ly/61MvPx>

Click **View Event Recordings** and choose *Converting Unstructured FrameMaker to DITA*; the **Mif2Go** part of the presentation starts at 33:15.

You can also view a YouTube video created by Yves Barbion that shows how to get image attributes from FrameMaker into DITA via **Mif2Go**:

http://www.youtube.com/watch?v=VAuQwEn_ogw

15.1.2 Understanding what you need to know about DITA

To use **Mif2Go** effectively to produce DITA output, you need a basic knowledge of DITA, from study of other materials. Teaching our customers DITA is beyond the scope of the **Mif2Go User's Guide**. You have to know *what* you want; then perhaps we can tell you *how* to make it happen with **Mif2Go**.

If you are not familiar with DITA, here are some good starting points and reference sites:

IBM *Introduction to the Darwin Information Typing Architecture*:

<http://www-128.ibm.com/developerworks/xml/library/x-dita1/index.html>

Online community for the Darwin Information Typing Architecture OASIS Standard:

<http://dita.xml.org/>

OASIS *DITA Architectural Specification*:

<http://docs.oasis-open.org/dita/v1.1/CD01/overview/overview.html>

Cover pages on DITA:

<http://xml.coverpages.org/dita.html>

DITA World *Comprehensive List of DITA Resources*:

<http://www.ditaworld.com/>

On-line DITA lists:

<http://tech.groups.yahoo.com/group/framemaker-dita/>

<http://tech.groups.yahoo.com/group/dita-users/>

Comtech Services book, available at <http://www.comtech-serv.com/dita.shtml#book>:

Linton, Jen and Bruski, Kylene (2006). *Introduction to DITA: A Basic User Guide to the Darwin Information Typing Architecture*. Denver, CO: Comtech Services

15.1.3 Clarifying your purpose for creating DITA output

Mif2Go supports two general purposes for creating DITA output from FrameMaker:

[Migrate legacy content to DITA XML](#)

[Export current content to DITA as needed.](#)

A third potential purpose might be to use DITA as an intermediate step in converting documents from unstructured to structured FrameMaker. You could use **Mif2Go** to produce DITA XML from your unstructured files, then bring the results back into structured FrameMaker. This should be a lot faster than developing FrameMaker conversion tables.

*Migrate legacy
content to DITA
XML*

When you migrate legacy content from FrameMaker to DITA XML, completeness is less important than it would be if you retain source in FrameMaker. After converting your document you edit in an XML environment, so (for example) you can add metadata after conversion. Even validity can be relaxed, if your existing document does not quite

measure up. As long as the XML is well formed, you can use XSLT to make adjustments. You can even run XSLT from within **Mif2Go**, with a command that invokes, for example, Saxon. See §34.4 [Executing operating-system commands](#) on page 937.

If you are just beginning a proposed migration, you might want to stay with FrameMaker until you are satisfied with the results of converting *all* your FrameMaker documents to DITA. Keep editing in FrameMaker, and see how the DITA comes out. Make your FrameMaker documents conform to DITA architecture, if you can; you are sure to need specializations, which **Mif2Go** handles. This way you can continue to produce decent deliverables all the way through the process. At the same time, you can test alternative production workflows with the DITA versions you generate. And if in the end you do not think you can live with the strictures that DITA enforces, you can walk away without having a disaster to clean up.

The simplest route is to go from *unstructured* FrameMaker direct to DITA version 1.1, then import the DITA files into your next application. There would not be much point to bringing the files back into *structured* FrameMaker if you do not plan to keep using FrameMaker. In fact, it would be a bad idea, because structured FrameMaker (version 8) currently supports only DITA version 1.0, so you would lose index features, graphics scaling, and so forth. Unstructured FrameMaker does allow DITA version 1.1 features natively, and **Mif2Go** output to DITA version 1.1 retains those features.

For a list of features added in DITA version 1.1 that would be lost if you went back to DITA version 1.0, see:

http://wiki.oasis-open.org/dita/DITA_1.1_Impact_Assessments

*Export current
content to DITA
as needed*

To continue using FrameMaker as source and export content to DITA as needed, you must interpolate into the DITA output any data required by DITA but not needed in FrameMaker. You can use FrameMaker markers or dedicated conditional paragraph formats for file-specific data, and **Mif2Go** configuration settings for general data items such as book revision level. You do not need XSLT for this purpose. In fact, you should not need XSLT at all, unless your FrameMaker document does not follow the same sequence of items that DITA expects. This is not likely, because DITA pretty much codifies what writers do anyway as good practice.

If you continue using FrameMaker as source, consider two possible ways to proceed:

- **DITA as an accommodation to others.** For this purpose you would want minimum disturbance to FrameMaker files; for example, you would keep multi-topic chapters, and chunk them out using **Mif2Go**.
- **DITA as an authoring model.** For this purpose you would make your FrameMaker files single-topic. Chapters would take on the role of ditamaps, and would import the topic files as insets.

If you are converting a FrameMaker book, the book file becomes a DITA map either way.

If you create content in DITA XML, then use the DITA Open Toolkit to generate other outputs, you might have difficulty producing some Help formats. Also, PDF output can look ugly. With **Mif2Go** you can continue to write in FrameMaker, and get a matching DITA set any time you need one. And you can produce DITA output from unstructured as well as from structured FrameMaker.

15.1.4 Converting from structured vs. unstructured FrameMaker

You do not have to use structured FrameMaker to produce DITA XML with **Mif2Go**. You *can* use structured FrameMaker, *provided your EDD specifies distinct formats for the elements*, rather than the Word-style formatting some structured-FrameMaker users prefer.

Whether you use structured or unstructured FrameMaker, to produce completely valid DITA output, the content of your FrameMaker document must be arranged to fit the DITA architecture.

To use structured FrameMaker as a DITA authoring tool, you need DITA-FMx:

<http://leximation.com/dita-fmx/>

This is true even if you are using FrameMaker version 9, which supports DITA 1.1.

15.1.5 Understanding what information you must supply

Mif2Go support for DITA requires you to supply the following kinds of information in addition to your FrameMaker document:

[DTD properties](#)

[FrameMaker mappings](#)

[Disambiguation.](#)

<i>DTD properties</i>	Mif2Go provides built-in content models for basic DITA topic types. If you are using structured FrameMaker, you can abstract content-model information from your DITA DTD. To add or replace a content model, use free command-line utility dtdd2ini to generate a content model from a DTD, and produce a content-model configuration file for your DITA project. See §32.2.2 Generating a content model from a DTD on page 906.
<i>FrameMaker mappings</i>	For either structured or unstructured FrameMaker, you must map FrameMaker formats to the content model in terms of element names, element parentage, required level (if any), inline/block nature, and any requirement to be first under any parent. This information goes into your project configuration file, and possibly into chapter-specific configuration files. You might have to use markers in your FrameMaker document to provide information such as topic IDs, element names, and attributes, in cases where these items cannot be derived from the document.
<i>Disambiguation</i>	In an unstructured FrameMaker document, presentation might be the same for several different usages. Mif2Go cannot necessarily determine whether (for example) text tagged <code><Italic></code> is a computer term, a foreign language term, or a long quote, all of which have different representations in DITA, based on the context. The onus is on the author to disambiguate these usages, if necessary by inserting DITA markers in individual instances of particular formats. Mif2Go does handle a few presentational features automatically; for example, by default forced returns (FrameMaker Shift+Enter) are converted to spaces.

15.1.6 Understanding how Mif2Go generates DITA output

Mif2Go creates DITA XML for the following, based on either structured or unstructured FrameMaker chapter (.fm) and book (.book) files (*not* FrameMaker XML files):

[DITA elements](#)

[DITA topics](#)

[DITA links](#)

[DITA index terms](#)

[DITA maps](#)

[DITA specializations and constraints](#)

DITA elements **Mif2Go** uses two methods to identify DITA element boundaries in a FrameMaker document:

- FrameMaker formats, mapped to DITA elements in your project configuration file; see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487

- FrameMaker markers in your document that identify elements where mapping is not feasible, either because the same format is used for several elements, or because character formats would have to nest for DITA phrases; see §15.15 [Overriding DITA settings with markers](#) on page 536.

Mif2Go determines element nesting (which is not explicit in unstructured FrameMaker) according to settings in the configuration file. The `<body>` wrapper starts after any metadata, and continues until the end of the topic or until an element mapped to related links appears. **Mif2Go** provides the `<body>` and `<related-links>` wrappers. **Mif2Go** wraps each image that has a title in a `<fig>` element, along with the title.

DITA topics **Mif2Go** supports concept, task, reference, topic, glossary (for DITA 1.1), map, bookmap (for DITA 1.1), and custom topic types. You can specify the topic type in a **DITATopic** marker. If a FrameMaker file contains only a single type of topic, you can set a default per file; then you would need markers only for exceptions. A format mapped to the `<title>` element (at level 1) identifies the start of a topic. A topic ends at the start of the next topic, or at the end of the file.

DITA links **Mif2Go** generates DITA `<xref>` elements from FrameMaker cross references and hypertext links. For links to target elements that do not already have IDs specified via **DITAElemID** markers, **Mif2Go** uses either the **newlink** marker content or the FrameMaker cross-reference numeric ID for the element ID. If a target element contains both a **newlink** marker and a cross-reference marker, the **newlink** marker content becomes the element ID. If a target element has multiple **newlink** markers, the content of the first **newlink** marker becomes the element ID.

DITA index terms **Mif2Go** moves all index entries in a paragraph to the end, just before the closing tag of the paragraph element. When that element is not valid as a container for `<indexterm>`, as is the case with `<title>`, **Mif2Go** wraps the index entries in a `<ph>` element, which is valid in many more elements (including `<title>`). If the `<ph>` wrapper is not valid in that location, it is up to the author to move the index marker in FrameMaker to a place where the resulting `<indexterm>` (or, if necessary, its `<ph>` wrapper) will be valid in DITA.

DITA maps **Mif2Go** generates a DITA map for each FrameMaker chapter file, and also for the FrameMaker book. The book map can include all chapter content, or just reference the chapter maps. DITA `<reltable>` elements are produced from ALinks; each distinct subject creates one `<relrow>`. The topic types are retained from the topics themselves. The linking attribute is set to target or to source in **DITARelLinking** markers in the topics. See §16 [Configuring DITA maps](#) on page 539.

DITA specializations and constraints **Mif2Go** provides a simplified way to represent the hierarchical information that is contained in the DTD (and in `.mod` files) for specialized topic types. **Mif2Go** does not include class attributes in DITA output; those are best left to the DTD to provide.

For DITA 1.2 you can impose a “no pernicious mixed content” constraint by using constraint modules, without specializing. You can produce constraints the same way as a specialization, using **Mif2Go** utility `dt2ini.exe` on the local document type shell and referencing the result in your project configuration file. (This one-time extra step is required as a result of licensing.)

To include specializations or constraints in DITA output, see §32 [Working with content models](#) on page 905.

15.1.7 Creating valid DITA XML output

Mif2Go does not try to validate the output from a DITA conversion; you must use a validating parser to check output validity. However, **Mif2Go** does ensure valid parental

relationships and first-child restrictions. Valid sequence of items within those constraints has to come from the implied or explicit structure of the FrameMaker document.

Mif2Go does try to help you create valid DITA output. When **Mif2Go** finds that the next paragraph is mapped to an element that would not be valid at that point in DITA XML, **Mif2Go** looks for a way to make the element valid. **Mif2Go** might close existing elements, or interpolate new elements. If you do not like the result, you must tell **Mif2Go** (via configuration settings) what to interpolate instead. See §32.5 [Understanding how Mif2Go uses content models](#) on page 911.

15.2 Setting up a DITA XML project

When you set up a DITA XML project from within FrameMaker, if starting configuration file `_m2dita.ini` is not already present in the project directory, **Mif2Go** copies this configuration file to the project directory for you; see §4.3 [Understanding where project settings come from](#) on page 102.

To add or change any of the options described in this section, edit configuration file `_m2dita.ini`, located in your project directory. Or, to apply the changes to all of your DITA XML projects, edit the configuration file referenced by `_m2dita.ini`:

```
%omsyshome%\m2g\local\config\local_m2dita_config.ini.
```

See §30.5 [Deciding which configuration file to edit](#) on page 856.

In this section:

- §15.2.1 [Creating a DITA XML project](#) on page 478
- §15.2.2 [Choosing set-up options for a DITA XML project](#) on page 479
- §15.2.3 [Specifying DITA output options](#) on page 480
- §15.2.4 [Specifying DITA version](#) on page 480
- §15.2.5 [Configuring the DITA DTD SYSTEM identifier](#) on page 481
- §15.2.6 [Ensuring FrameMaker 8 import compatibility](#) on page 481
- §15.2.7 [Substituting document format names for default names](#) on page 481

15.2.1 Creating a DITA XML project

To create a DITA XML project:

1. Create a directory for DITA output, separate from the directory where your FrameMaker document is located.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
3. Name your DITA XML project, and browse to the project directory you created in [Step 1](#).
4. Choose output type DITA XML and click **OK**.
5. Check options in the *Set Up DITA Project* dialog (see §15.2.2 [Choosing set-up options for a DITA XML project](#) on page 479).
6. Click **OK** to dismiss the dialog.

When you click **OK** on the *Set Up DITA Project* dialog, **Mif2Go** copies a new project configuration file, `_m2dita.ini`, to your project directory. In addition to the settings you specified in the set-up dialog, this file contains a series of empty configuration sections. *It is up to you to fill these sections with the rest of the settings required to convert your*

document. Use a text editor to edit `_m2dita.ini`; see §4.1 [Working with Mif2Go configuration files](#) on page 91. Pay special attention to sections `[DITAParaTags]`, `[DITACHarTags]`, and `[DITAParents]`; see §15.2.7 [Substituting document format names for default names](#) on page 481.

Note: Your FrameMaker book name must not duplicate the name of any chapter file.

15.2.2 Choosing set-up options for a DITA XML project

When you choose DITA as the output type for a new project, the *Set Up* dialog shown in [Figure 15-1](#) opens. [Table 15-1](#) shows the corresponding settings in the configuration file. *You must edit the configuration file to specify additional options.*

See also:

§3.4 [Choosing project set-up options](#) on page 79

§13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425

Figure 15-1 Set Up DITA Project

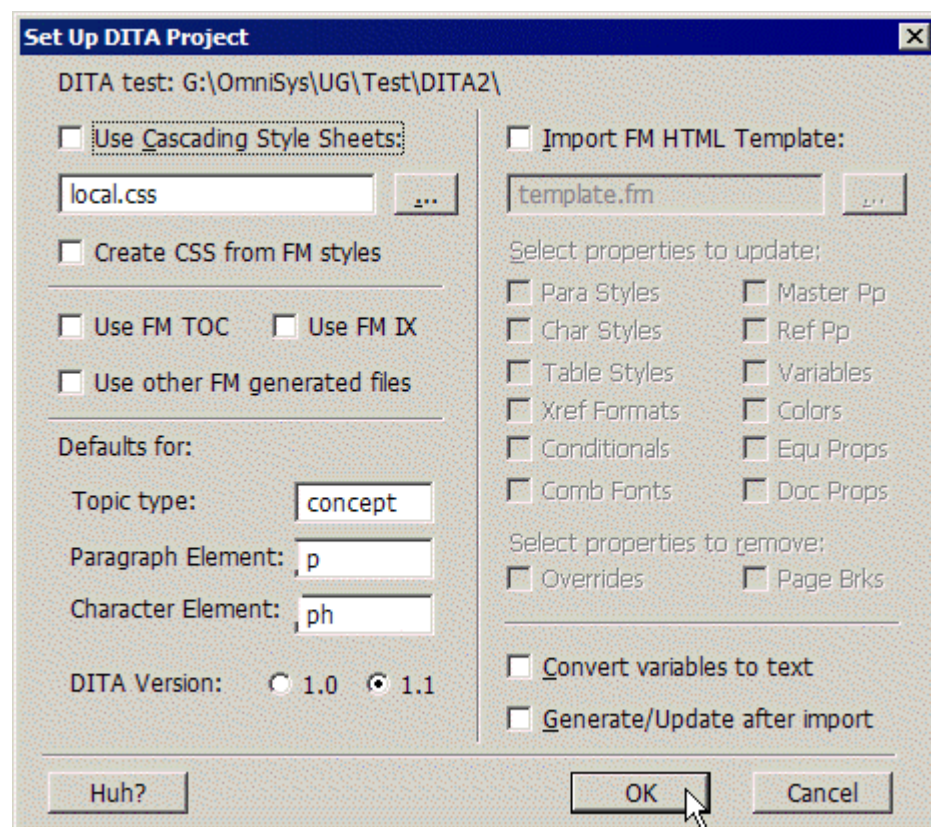


Table 15-1 DITA set-up options and configuration settings

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Use Cascading Style Sheets	[CSS]	UseCSS=Yes	Yes	22.4
Path to CSS file	[CSS]	CssFileName=mycss.css	local.css	22.4.3
Create CSS from FM styles	[CSS]	WriteCssStylesheet=Once	Once	22.4.3
Defaults for:				

Table 15-1 DITA set-up options and configuration settings (continued)

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Topic type	[DITAOptions]	DefTopic=topic	concept	15.9.2
Paragraph Element	[DITAOptions]	DefParaElem=element	p	15.4.3.3
Character Element	[DITAOptions]	DefCharElem=element	ph	15.4.4.3
DITA version	[DITAOptions]	DITAVer=N	1	15.2.4

15.2.3 Specifying DITA output options

To change the file extension for DITA output (not recommended):

```
[Setup]
FileSuffix = .ext
```

The default file extension is `.dita`. Unless you have a compelling reason, use `.dita`, not `.xml`, for the output file extension. Many places in DITA depend on that extension, and will break if you do not use it; for example, related links would be assigned the wrong `@format` value, and so would not work correctly in many tools.

The `[Setup]` setting is used by the **Mif2Go** plug-in. For this setting to take effect, you might have to close FrameMaker, make the change, save your configuration file, and then re-open FrameMaker.

If you have FrameScript installed on your system, you can use a script to create more “human readable” topic names for DITA output. See:

<http://frameautomation.com/2010/03/24/renaming-dita-map-topics/>

15.2.4 Specifying DITA version

By default, **Mif2Go** produces output for DITA version 1.1. However, you can restrict output to features that conform to DITA version 1.0, and you can generate version 1.2 output by specifying 1.1 and using the mechanism provided for specializations and content models; see §32 [Working with content models](#) on page 905.

To specify DITA version 1.0 output:

```
[DITAOptions]
; DITAVer = DITA version point number, 1 (default, for 1.1)
; or 0 (for 1.0)
DITAVer = 0
```

When `DITAVer=1`, output can include the following DITA 1.1 features:

- elements `<index-see>`, `<index-see-also>`, and `<index-sort-as>`
- index range attributes
- elements `<abstract>`, `<foreign>`, `<unknown>`, and `<data>`
- topic types `glossentry` and `bookmap`, and the elements they contain.

When `DITAVer=0`, these features are omitted from output.

We recommend sticking with DITA version 1.1. However, FrameMaker 7.2 and 8.0 do not support DITA version 1.1; if you expect to bring your DITA 1.1 output back into FrameMaker 7.2 or 8.0 with these features intact, you will need the Leximation DITA-FMx plug-in:

<http://www.leximation.com/dita-fmx/>

Note: When [DITAOptions]FM8Import=Yes, **Mif2Go** changes the default value of DITAVer to 0; see §15.2.6 [Ensuring FrameMaker 8 import compatibility](#) on page 481.

15.2.5 Configuring the DITA DTD SYSTEM identifier

To have **Mif2Go** use a DTD SYSTEM identifier without a path, when required for other DITA processing applications:

```
[DITAOptions]
; UseDTDPath = Yes (default, use full URL to DTD at OASIS) or No (just
; use the DTD name with no path, required for XMetaL and some CMSs).
UseDTDPath = No
```

15.2.6 Ensuring FrameMaker 8 import compatibility

If you are using FrameMaker version 8, and you expect to use maps or to round-trip your DITA output, specify the following option:

```
[DITAOptions]
; FM8Import = No (default) or Yes (restrict output for compatibility
; with FrameMaker Version 8 DITA import and map usage).
FM8Import = Yes
```

You do *not* need this setting if you are using the Leximation DITA-FMx plug-in with FrameMaker version 8; see:

<http://www.leximation.com/dita-fmx/>

When FM8Import=Yes, **Mif2Go** changes default values for the following settings:

<u>[DITAOptions] keyword</u>	<u>FM8Import default</u>	<u>Consequence</u>	<u>Ref.</u>
DITAVer	0	Output is restricted to DITA version 1.0 features	15.2.3
MapTopicID	No	Topic IDs are omitted from map references	16.2.1.7
MapTopicmeta	No	The <topicmeta> element is excluded from maps	16.2.1.8
WrapTopicFiles	Yes	Each topic file is wrapped in a <dita> element	15.8.5
UsePtSuffix	Yes	Image width and height attributes are in points	15.7.7

See also:

§16.2.1.2.2 [Configuring chapter maps for FrameMaker 8 import](#) on page 540.

15.2.7 Substituting document format names for default names

When **Mif2Go** creates a configuration file for a new DITA project, the default settings in all configuration sections that involve assigning a property to a FrameMaker format are based on the format names in the standard template that comes with FrameMaker, not on the format names in your document.

If your document uses format names different from those in the standard FrameMaker template, *you must change these settings* to assign properties to the corresponding names in your document instead. You must also add settings for other formats in your document that map to other than the default block and inline elements, which are <p> and <ph>, respectively.

Note: It is best not to delete any default settings until you know what you are doing.

Default settings in the following configuration sections are based on formats in the standard FrameMaker template:

```
[DITAParaTags]
[DITACHarTags]
[DITACHarTypographics]
[DITALEvels]
[DITAParents]
```

[DITAParaTags] Only a few paragraph formats are automatically mapped to DITA elements:

```
[DITAParaTags]
; Frame para format (wildcards OK) = DITA element
Body=p
Heading*=title
Numbered=li
Numbered1=li
Bulleted=li
FigureTitle=title
CellBody=p
CellHeading=p
```

See §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487.

[DITACHarTags] Only one character format is automatically mapped to a DITA element:

```
[DITACHarTags]
Emphasis=i
```

See §15.4.4 [Mapping character formats to DITA inline elements](#) on page 492.

[DITACHarTypographics] The only character format automatically included here is *Emphasis*; combined with its default mapping in [DITACHarTags], this setting makes *Emphasis* both bold and italic:

```
[DITACHarTypographics]
; Frame char format (wildcards OK) = DITA typographic
; elements (any or all of b, i, u, tt, sup, or sub) to use in
; addition to the element to which the format is mapped in
; [DITACHarTags].
Emphasis=b
```

See §15.4.5 [Assigning multiple typographic elements to a format](#) on page 494.

[DITALEvels] By default, a paragraph format named *Title* that is not explicitly mapped in [DITAParaTags] becomes a <title> element, which is always at level 1 in its DITA topic:

```
[DITALEvels]
; Frame format (para or char, wildcards OK) = level in DITA (not
; Frame) file required for the DITATag specified for this element.
Title=1
Heading*=3
```

See §15.5.13 [Specifying DITA element levels](#) on page 509.

[DITAParents] Only a few paragraph formats are automatically assigned DITA parents:

```
[DITAParents]
; Frame format (para or char, wildcards OK) = required parents
Title=topic
Heading*=section
Numbered1=ol
Numbered=ol
Bulleted=ul
FigureTitle=fig
```

See §15.5.2 [Designating DITA ancestor elements](#) on page 502.

15.3 Specifying general options for DITA

This section summarizes DITA-specific default values and recommended options for configuration settings in the following areas:

[Declaration](#)
[Standard XML options](#)
[Filtering options](#)
[Style options](#)
[Indexing](#)
[CSS](#)
[Context-sensitive help](#)
[Assembling files for distribution](#)

Declaration **Mif2Go** sets the following default values for the PUBLIC declaration, depending on the DITA version and on the topic type. For example, for DITA version 1.1 and topic type concept:

```
[HTMLOptions]
HTMLDocType="-//OASIS//DTD DITA 1.1 Concept//EN"
HTMLDTD="docs.oasis-open.org/dita/v1.1/CS01/dtd/concept.dtd"
```

If you need the declaration to comply with the requirements of third-party tools, you can override the default values. See §32.7.2 [Overriding settings in a DITA content model](#) on page 914 and §32.7.4 [Overriding declarations in a DITA map content model](#) on page 915.

Standard XML options The following XML settings cannot be overridden:

```
[HTMLOptions]
AllowOverrides = No
AlignAttributes = No
NoFonts = Yes

[Graphics]
GraphScale = Yes

[Tables]
TableAttributes = No
CellAlignAttributes = No
CellColorAttributes = No
```

Filtering options These settings provide DITA-specific default values for assorted options; you do not have to include the following settings in your configuration file unless you change their values:

```
[HTMLOptions]
; The following are the DITA-specific defaults for each setting:
FileSuffix = .dita
RunInHeads = Normal
RemoveANames = Yes
ATagElement = xref
XMLEncoding = UTF-8
NumericCharRefs = No
FootInlineTag = fn
HardRetPara = No
RemoveEmptyParagraphs = Yes
RemoveEmptyTableParagraphs = Yes
```

If your document uses run-in headings only for presentational purposes, you might want to set RunInHeads=Runin; see §21.3.2 [Converting sidehead and run-in paragraph formats](#) on page 648.

See §13 [Converting to HTML/XHTML](#) on page 423.

Style options Keep empty paragraphs empty (if not removed):

```
[HTMLOptions]
EmptyParaContent =
```

See §21.3.9 [Providing content for empty paragraphs](#) on page 651.

Footnotes are DITA-specific, but you can remove them:

```
[HTMLOptions]
Footnotes = None
```

A footnote that appears in a <title> element will be wrapped in a <ph> element. If you are using the DITA Open Toolkit, see §15.10.5 [Omitting <xref> elements from footnotes](#) on page 529.

By default, **Mif2Go** suppresses FrameMaker autonumbers for DITA:

```
[HTMLParaStyles]
* = NoAnum
```

Autonumbers violate DITA architecture. However, if you use FrameMaker numbering properties for purposes that have nothing to do with numbering, you can override the suppression for selected paragraph formats:

```
[HTMLParaStyles]
ParaFmt = Anum
```

If you want a graphic in DITA, you reference it, then add it when you render output from DITA. The graphic does not belong in the DITA itself, so if it is present in FrameMaker you need to mark it for deletion. How you do that depends on how the graphic was added in FrameMaker. For example, if the graphic is a *Frame Above*, set:

```
[HTMLOptions]
RemoveFramesAbove=Yes
```

This removes all such frames, which is probably what you want; see §21.3.7 [Keeping or removing reference frames](#) on page 651.

If the graphic is anchored in the text, apply a condition, and exclude that condition when converting to DITA.

By default, **Mif2Go** converts forced returns (FrameMaker **Shift+Enter**) to spaces for DITA. To simply close and reopen the paragraph tag (without attributes) instead:

```
[HTMLOptions]
XMLBreakPara = Yes
```

To do so selectively by paragraph format:

```
[HTMLParaStyles]
ParaFmt = XMLBreak
```

See §14.4.5 [Configuring forced returns for XML](#) on page 465.

To preserve line or page breaks in DITA, see §15.4.8 [Including PIs for line, column, or page breaks](#) on page 499.

Do not assign the [HTMLParaStyles] Split property to any FrameMaker format. For DITA, **Mif2Go** splits files according to topic starts; see §15.8.2 [Splitting FrameMaker files into DITA topic files](#) on page 520.

By default, for DITA XML output **Mif2Go** adds a px suffix to width and height attribute values for images sized in pixels. To omit the suffix:

```
[Graphics]
UsePxSuffix = No
```


See §23.9.5 [Specifying px units for graphics sized in pixels](#) on page 722, and §15.7.7 [Understanding why images might look incorrectly scaled](#) on page 519.

Indexing For DITA version 1.1 **Mif2Go** uses FrameMaker sort strings to produce `index-sort-as` elements, so you might want to set the following Help option:

```
[Index]
UseSortString = Yes
```

See §7.5.9.4 [Choosing whether to use FrameMaker index sort strings](#) on page 218, and §15.4.10 [Converting index markers to <indexterm> elements](#) on page 500.

CSS The CSS file **Mif2Go** generates for DITA specifies classes only, no tags, so that it can be used for HTML outputs generated from the DITA files. These options are in effect by default:

```
[CSS]
WriteClassAttributes = No
ClassIsTag = No
```

which results in `[DITAOptions]UseOutputClass=No`.

To include CSS class names, specify `[DITAOptions]UseOutputClass=Yes`, then convert `@outputclass` to class attributes in the HTML; see §15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498. Setting `[CSS]UseCSS=Yes` also sets `[DITAOptions]UseOutputClass=Yes`; see §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684.

Note: To include CSS class names as `outputclass` attributes, make sure your configuration file does *not* specify `[CSS]WriteClassAttributes=No`.

By default, **Mif2Go** uses FrameMaker cross-reference format names and link character format names to set the `@outputclass` for cross references and hypertext links:

```
[CSS]
XrefFormatIsXrefClass = Yes
```

See §22.7.8 [Using link format names as CSS class names](#) on page 696.

Context-sensitive help By default, DITA output includes all context-sensitive help targets provided in your source document via **TopicAlias** markers, in the following form:

```
<data name="topicalias" value="IDH_about" />
```

To exclude these targets from DITA output:

```
[DITAOptions]
; UseTopicAlias = Yes (default, include CSH targets in DITA output)
; or No
UseTopicAlias=No
```

See:

§7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239

§15.14 [Including CSH targets in DITA XML](#) on page 535

The `id` attribute has no use in DITA maps, so is not essential.

Assembling files for distribution These settings provide DITA-specific default values for gathering converted files for further processing; you do not have to include these settings in your configuration file if the following default values are satisfactory:

```
[Automation]
; The following are the DITA-specific defaults for each setting:
WrapCopyFiles = *.dita *.ditamap *.bookmap *.dtd *.mod *.ent *.xsd
CssCopyFiles = *.css *.xsl
GraphCopyFiles = *.gif *.jpg *.png *.svg
EmptyOutputFiles = *.dita *.ditamap *.bookmap *.ref *.grx
```

See §35 [Producing deliverable results](#) on page 955.

15.4 Configuring DITA elements

You must ensure that all elements used within the body of a topic are valid for the specified topic type.

In this section:

- §15.4.1 [Understanding how Mif2Go delimits DITA elements](#) on page 486
- §15.4.2 [Treating FrameMaker format names as DITA element names](#) on page 486
- §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487
- §15.4.4 [Mapping character formats to DITA inline elements](#) on page 492
- §15.4.5 [Assigning multiple typographic elements to a format](#) on page 494
- §15.4.6 [Assigning attributes to DITA elements](#) on page 495
- §15.4.7 [Preserving whitespace in block elements](#) on page 499
- §15.4.8 [Including PIs for line, column, or page breaks](#) on page 499
- §15.4.9 [Providing a <shortdesc> element for a DITA topic](#) on page 500
- §15.4.10 [Converting index markers to <indexterm> elements](#) on page 500

See also:

- §15.6 [Converting tables to DITA XML](#) on page 510

15.4.1 Understanding how Mif2Go delimits DITA elements

Mif2Go closes each element mapped from a FrameMaker paragraph format when a paragraph in that format ends. For example, even though a DITA list can be inside a `<p>` element, **Mif2Go** does not put it there; instead, the `<sl>` follows the `<p>`. Only elements that are marked as inline, including elements mapped from FrameMaker character formats, and inline images, are placed within a `<p>` element.

An interpolated block element stays open until **Mif2Go** encounters a paragraph that is not valid in that block.

Lists are identified by the FrameMaker format mapped to the list element it populates, such as `` or `<sl>`; see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487, or by the parent of the mapped element; see §15.5.2 [Designating DITA ancestor elements](#) on page 502. **Mif2Go** provides the wrappers around the list items and around the whole list.

To minimize the need for markers in your FrameMaker document it is good practice to use distinct FrameMaker format names to identify different kinds of lists (*List*, *Bulleted*, *Numbered*, *ParamTerm*), body paragraphs (*Body*, *Example*), character formats (*Strong*, *Emphasis*), and so forth.

The easiest way to migrate and apply paragraph and character formatting from your FrameMaker document is to use the DITA `outputclass` attribute, which **Mif2Go** will set for you, to reference your current Framemaker formats or any others you map to those formats in `[ParaClasses]` or `[CharClasses]`. See §15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498.

15.4.2 Treating FrameMaker format names as DITA element names

If most of your FrameMaker formats are named for DITA elements, you can lessen the chore of mapping formats to elements by directing **Mif2Go** to use the format name as the

DITA element name wherever possible (that is, where the content model includes an element of that name). This works only if the named element is of an appropriate type: block allowing text for a paragraph format, or inline allowing text for a character format; see §32.6 [Inspecting and correcting element types](#) on page 912.

However, unless your FrameMaker template is specifically designed for DITA, leaving any paragraph format unmapped is risky; some formats might match the names of DITA elements that do not do what you want.

To map FrameMaker format names to DITA elements of the same name where possible:

```
[DITAOptions]
; UseFormatAsTag = No (default, if tag unmapped use default elem),
; or Yes (if unmapped, use Frame format name if valid in content
; model).
UseFormatAsTag = Yes
```

When `UseFormatAsTag=Yes`, any FrameMaker format with a name that is the same as a DITA element name in the current content model is mapped to that element.

When `UseFormatAsTag=No`, unmapped format names that do not correspond to appropriate DITA element names are mapped to the default element; see:

§15.4.3.3 [Specifying a default element for unmapped paragraph formats](#) on page 489

§15.4.4.3 [Specifying a default element for unmapped character formats](#) on page 494.

15.4.3 Mapping paragraph formats to DITA block elements

Paragraph formats must be mapped to DITA block elements that can contain text, not to inline elements or topic containers. The end of a FrameMaker paragraph always ends the block element to which it is mapped.

Make sure target elements can contain text

When you map paragraph formats to DITA block elements, you must ensure that the element mapped to is allowed to contain text. For example, in a `<task>`, do not map to `<step>`; map to `<cmd>` or `<info>`, which fit inside `<step>`. For list items that can include more than one paragraph, map the paragraph format(s) to `<p>`, and designate their including list element; see §15.5 [Nesting DITA block elements](#) on page 501.

In this section:

§15.4.3.1 [Assigning DITA elements to FrameMaker paragraph formats](#) on page 487

§15.4.3.2 [Omitting element tags for selected paragraph formats](#) on page 488

§15.4.3.3 [Specifying a default element for unmapped paragraph formats](#) on page 489

§15.4.3.4 [Omitting invalid tags for default DITA block elements](#) on page 489

§15.4.3.5 [Overriding element mapping for paragraph formats](#) on page 490

§15.4.3.6 [Providing aliases for paragraph formats](#) on page 490

§15.4.3.7 [Mapping paragraph format aliases to different elements](#) on page 491

§15.4.3.8 [Mapping paragraph format aliases algorithmically](#) on page 491

§15.4.3.9 [Mapping several paragraphs formats to the same element](#) on page 492

15.4.3.1 Assigning DITA elements to FrameMaker paragraph formats

To map paragraph formats in your document to DITA elements, assign the element name to the format name:

```
[DITAParaTags]
; Frame paragraph format (wildcards OK) = DITA element, can be
; overridden by a DITATag marker; or Frame format = No.
ParaFmtName = elementname
```

For example:

```
[DITAParaTags]
Heading* = title
Meta = keyword
Body = p
Example = p
List = sli
Numbered1 = p
Numbered = p
Bulleted = p
DefTerm = dt
DefDescription = dd
ParamTerm = pt
ParamDescription = pd
TableTitle = title
CellHeading = p
CellBody = p
Figure Title = title
Step = cmd
Syntax = p
CellContent = No
GlossItem = glossterm
```

<i>Default element</i>	The default element for a FrameMaker paragraph format that is not mapped in [DITAParaTags] depends on the value of [DITAOptions]UseFormatAsTag; see §15.4.2 Treating FrameMaker format names as DITA element names on page 486.
<i>Do not map to footnote or table elements</i>	Mif2Go processes footnotes and table components separately; do not map any paragraph formats to footnote elements, or to any table component (table, title, row, or cell) elements. See §15.6 Converting tables to DITA XML on page 510. If you are using the DITA Open Toolkit, see §15.10.5 Omitting <xref> elements from footnotes on page 529.
<i>Do not map to element sets</i>	You can assign element sets in [DITAParents] and in [DITAFirst], but you cannot use them for tags in [DITAParaTags]. See §15.5.5 Specifying alternate ancestries for the same element on page 504.
<i>Specify ancestry for list formats</i>	For list formats, if mapping the format to an element is not sufficient to identify the list type, you must also specify the parent of the element; see §15.5.2 Designating DITA ancestor elements on page 502. Definition lists can be derived from paragraph pairs, possibly with run-in headings for the term.
<i>Add typographic elements</i>	To add typographic elements (b, i, u, tt, sup, or sub) in addition to the element to which a FrameMaker format is mapped, see §15.4.5 Assigning multiple typographic elements to a format on page 494.
<i>Manage forced returns</i>	Forced returns in FrameMaker have no counterpart in DITA XML. Mif2Go replaces each forced return with a space. If you want something else to happen, use a marker to insert a processing instruction where each forced return occurs in your document; see §15.4.8 Including PIs for line, column, or page breaks on page 499.

15.4.3.2 Omitting element tags for selected paragraph formats

To specify that a particular FrameMaker paragraph format should *not* be mapped to any element:

```
[DITAParaTags]
ParaFmt = No
```

When *ParaFmt*=No, tags for the format are omitted from output, leaving the text of the paragraph inside the enclosing element. Compare this setting with the effect of the NoPara format property; see §21.3.6 [Stripping paragraph properties](#) on page 650. The

`ParaFmt=No` setting is similar, but for DITA output it is recognized in places where the `NoPara` property is not. If you do not get the correct result with one, try the other.

Mif2Go assumes a paragraph mapped to `No` contains `PCDATA`, and checks to ensure `PCDATA` is valid at the current point. If a paragraph format mapped to `No` has no text content, **Mif2Go** ignores it, checking to see if `PCDATA` is valid only if there really is some `PCDATA`.

*Delete
paragraphs with
unwanted text*

If an instance of a paragraph format mapped to `No` contains text, and `PCDATA` is not valid in the current enclosing element, then if closing current tags does not solve the problem, **Mif2Go** does not try to interpolate. Instead, **Mif2Go** issues a “parent error”. In this case it is your responsibility to map such a paragraph format to an appropriate element rather than to `No`. For example, if you have a figure anchor paragraph that also contains text, you would want to map its format to something allowable within `<fig>`, such as `<desc>`. If you do not want the text to appear in DITA output, instead of mapping the paragraph format to `No` in `[DITAParaTags]`, mark the format for deletion. For example:

```
[HTMLParaStyles]
FigAnchor = Delete
```

See §21.3.12 [Eliminating unwanted paragraphs](#) on page 652.

*Map code-
example formats
to No*

You can map formats to `No` for code examples (which can run on for pages), to avoid having each line of code mapped to a separate `<codeblock>` element:

```
[DITAParaTags]
Code* = No

[DITAParents]
Code* = codeblock
```

In this example, specifying ancestry guarantees that **Mif2Go** will retain the original line breaks, instead of normalizing them as for HTML or XML. See §15.5.2 [Designating DITA ancestor elements](#) on page 502.

15.4.3.3 Specifying a default element for unmapped paragraph formats

To specify a default element to use for unmapped paragraph formats:

```
[DITAOptions]
; DefParaElem = element to use for Frame para formats that are
; not mapped in [DITAParaTags], default is "p".
DefParaElem = p
```

If your configuration file does not include a value for `DefParaElem`, **Mif2Go** uses one of the following as the element for an unmapped format: if `UseFormatAsTag=Yes` and the `FrameMaker` format name (adjusted as for CSS class names) matches the name of a valid element in the current content model, the format is mapped to that element; otherwise, the format is mapped to `p`, the default value of `DefParaElem`. See §15.4.2 [Treating FrameMaker format names as DITA element names](#) on page 486.

15.4.3.4 Omitting invalid tags for default DITA block elements

Some DITA block elements allow only `#PCDATA`, not paragraph tags. When a “normal” paragraph must be placed inside one of these blocks, the paragraph tag should be omitted.

If some paragraph formats in your document are left unmapped, or are explicitly mapped to the default block element (usually `<p>`), the presence of such paragraphs in contexts where the default block element would not be valid could trigger unwanted interpolation of an arbitrary parent element. For example, if your `FrameMaker` document uses the standard *Body* paragraph format in table cells, and *Body* is either unmapped or is mapped

to <p>, for cells in a DITA properties table where <p> is not allowed, **Mif2Go** would wrap each instance of <p> in a parent element that is allowed.

For enclosing block elements that allow mixed content, you can avoid this problem by directing **Mif2Go** to omit the default paragraph tags instead of interpolating a parent.

To omit invalid default paragraph tags where mixed content is allowed:

```
[DITAOptions]
; DropInvalidParaTag = No (default) or Yes (if the para tag is the
; default DefParaElem <p> and is invalid, but #PCDATA is valid,
; drop the tag)
DropInvalidParaTag = Yes
```

See also:

§15.5.3 [Fixing up interpolated ancestries](#) on page 503

15.4.3.5 Overriding element mapping for paragraph formats

To override the element-name mapping for a given paragraph, insert a **DITATag** marker in the paragraph, with content the desired element name.

If mapping (or overriding mapping) does not suffice, and you do not need to specify a required ancestry for the element, use the following instead:

- [HTMLParaStyles] CodeBefore and CodeAfter properties for the format
- [ParaStyleCodeBefore] and [ParaStyleCodeAfter] sections to specify the element tags to surround the text.

See §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

Another alternative would be to bracket the text with **Config** markers, with content such as [ParaStyleCodeBefore]=<element> and [ParaStyleCodeAfter]=</element>; see §33.2.2 [Overriding settings with configuration markers](#) on page 921.

Note: Mappings provided via [ParaStyleCode*] settings or markers do not participate in any ancestry you specify for the element in question; see §15.5 [Nesting DITA block elements](#) on page 501.

15.4.3.6 Providing aliases for paragraph formats

If you are generating DITA from an unstructured FrameMaker document, your document might use the same format for different purposes, each purpose requiring that format to be mapped to a different DITA element, or to be nested in a different hierarchy, or both; or you might have several formats that map to the same DITA element. Because DITA uses semantic tags, whereas FrameMaker uses presentational tags, in some cases you need alternate names for paragraph formats to clarify semantic use cases.

To specify an alternate name, or alias, for a paragraph format:

```
[DITAAliases]
; Frame paragraph format = Frame format name to use in place of that
; paragraph format for DITA purposes, or Mif2Go selection macro.
ParaFmtName = AlternateName
```

An alias works in any [DITA*] configuration section that uses format names. The alias can be the name of another paragraph format in your document, provided the two formats map to exactly the same element with all the same DITA settings; or, the alias can be a name you invent.

For additional aliases for the same format, insert a **DITAAlias** marker in each instance of the format that requires a different alias, with content the name of another alias. You can also use a **DITAAlias** marker to override an alias assigned in section [DITAAliases].

You can use as many different aliases for the same paragraph format as your document requires. If you are creating new alias names, be careful not to duplicate the name of a format that is already in the FrameMaker paragraph catalog.

15.4.3.7 Mapping paragraph format aliases to different elements

Suppose your FrameMaker document includes a paragraph format named *Body2*, used in the following situations:

- most often as a continuation of a *Numbered1* or *Numbered* paragraph
- less often as a continuation of a *Bulleted* paragraph
- occasionally as a quotation, not part of any list.

This means that in different places in your document *Body2* would have to be mapped to different elements, or participate in different DITA hierarchies.

To resolve this conflict, you would assign aliases to the alternate uses of *Body2*. You could keep the original format name for the most frequent use; however, the name *Body2* does not convey anything about the differing semantics. Therefore you might want to use aliases for every use; for example, *Body2OList*, *Body2UList*, and *Body2Quote*.

To create an alias for the most prevalent use of *Body2*:

```
[DITAAliases]
Body2 = Body2OList
```

For the other two uses of *Body2*, you must insert a **DITAAlias** marker in each instance, with content one of the other aliases: *Body2UList* or *Body2Quote*. Then you could specify the following in configuration file `m2dita.ini`:

```
[DITAParaTags]
Body2?list = p
must = lq
```

Instead of using a **DITAAlias** marker, you can provide differential mappings of the same format by assigning **Mif2Go** macros to the aliases; see §28 [Working with macros](#) on page 787.

15.4.3.8 Mapping paragraph format aliases algorithmically

Suppose you use the same paragraph formats for numbered lists (for example, *NumFirst* and *NumNext*) both in material for `<concept>` topics and in material for `<task>` topics:

- When the list occurs in a `<concept>`, it should be mapped to `ol > li`.
- When the list occurs in a `<task>`, it should be mapped to `steps > step > cmd`.

To choose between alternate format aliases depending on the DITA context, you can assign to a FrameMaker format a **Mif2Go** macro that selects an alias according to the start tag of the current topic. For example:

```
[DITAAliases]
NumFirst = <$(($_ditastart is "task") ? "StepFirst" : "OLFirst")>
NumNext = <$(($_ditastart is "task") ? "StepNext" : "OLNext")>
```

These assignments say that if the topic start tag is `task`, use the *Step** format name, otherwise use the *OL** format name, in place of the original *Num** format name. The assignments use predefined macro variable `$_ditastart`, which contains the start tag of the current DITA topic.

You would assign the desired DITA elements to the alternate format names. For example:

```
[DITAParaTags]
Step* = step
OL* = li
```

You would also create entries for the alternate formats in [DITAParents], and if needed in [DITALevels], and so forth. In effect, you have created semantic formats from descriptive format.s.

The macros you assign do not have to select based on topic type; you can set macro variables to test for other properties or situations. The ability to assign a macro to a format name provides a general-purpose algorithmic way to map from FrameMaker formats to DITA elements, allowing you to deal with cases that normal mapping cannot handle.

15.4.3.9 Mapping several paragraphs formats to the same element

Suppose your FrameMaker document includes three different paragraph formats for quotations:

Quote in body text
FtnQ in footnotes
CellQ in table cells.

All three map to DITA element <lq>. You can make this semantic equivalence explicit in section [DITAAliases], and use the collective alias in other configuration sections:

```
[DITAAliases]
FtnQ = Quote
CellQ = Quote

[DITAParaTags]
Quote = lq
```

15.4.4 Mapping character formats to DITA inline elements

FrameMaker character formats must be mapped to DITA inline elements, not to block elements.

When you map character formats to DITA elements, make sure that the element mapped to is allowed to contain text. For example, do not map to <menucascade>, map to a <uicontrol> with <menucascade> as parent.

In this section:

- §15.4.4.1 [Assigning DITA elements to FrameMaker character formats](#) on page 492
- §15.4.4.2 [Including typographic elements in addition to mapped formats](#) on page 493
- §15.4.4.3 [Specifying a default element for unmapped character formats](#) on page 494
- §15.4.4.4 [Overriding element mapping for character formats](#) on page 494
- §15.4.4.5 [Using alternate character formats for menu cascades](#) on page 494

15.4.4.1 Assigning DITA elements to FrameMaker character formats

To map character formats in your document to DITA inline elements, assign the element name to the format name:

```
[DITACHarTags]
; Frame character format (wildcards OK) = DITA element, cannot be
; overridden by a DITATag marker; or Frame format = No.
CharFmtName = elementname
```

For example:

```
[DITACCharTags]
Strong=b
Emphasis=i
BoldItalic=b
Subscript=sub
Superscript=sup
Mono=tt
Link=No
```

To specify that a particular FrameMaker character format should *not* be mapped to an element:

```
[DITACCharTags]
CharFmtName = No
```

The value No means that the tags for the format should be omitted, leaving the text inside the enclosing element. For example, map the character formats you use for links and cross references to No. **Mif2Go** automatically generates <xref> tags from the cross references in your FrameMaker document, based on the format, but you do not need to map the format itself to any element. See §15.10 [Configuring cross references and links for DITA](#) on page 527.

The default element for a FrameMaker character format that is not mapped in [DITACCharTags] is the element designated by DefCharElem; see §15.4.4.3 [Specifying a default element for unmapped character formats](#) on page 494. It is best to map each character format to the most specific element possible, which is not often the default element.

15.4.4.2 Including typographic elements in addition to mapped formats

You can add typographic elements in addition to the element to which a FrameMaker character format is mapped.

To include typographic elements in DITA XML output:

```
[Typographics]
; UseTypographicElements = No (XML default, suppress b, i, u, tt, sub,
; and sup even when specified in a format) or Yes (HTML default)
UseTypographicElements = Yes
```

When UseTypographicElements=Yes, typographic elements b, i, u, tt, sup, and sub appear in DITA XML output *in addition to* any elements to which character formats are mapped. If your FrameMaker document includes change bars or overlines those are also represented as elements in DITA XML:

- Change bars become <chbar> elements; to convert them instead to attributes, see §15.4.6.7 [Converting FrameMaker change bars to condition attributes](#) on page 499.
- Overlines become <over> elements.

If any character formats are mapped to elements, and your project configuration file includes a UseTypographicElements setting, to avoid double nesting make sure you use the XML default value: UseTypographicElements=No.

Incorporate typographic elements sparingly, especially if you are using <outputclass>; see §15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498. DITA asks for semantic, not presentational, tags. It is best to let CSS handle the presentation later.

See also:

- §15.4.5 [Assigning multiple typographic elements to a format](#) on page 494
- §21.8 [Managing typographic elements for HTML or XML](#) on page 667

15.4.4.3 Specifying a default element for unmapped character formats

To specify a default element to use for unmapped character formats:

```
[DITAOptions]
; DefCharElem = element for Frame char formats that are not
; mapped in [DITACCharTags], default is "ph"
DefCharElem = ph
```

If your configuration file does not include a value for `DefCharElem`, **Mif2Go** uses one of the following as the element for an unmapped format: if `UseFormatAsTag=Yes` and the `FrameMaker` format name (adjusted as for CSS class names) matches the name of a valid element in the current content model, the format is mapped to that element; otherwise, the format is mapped to `ph`, the default value of `DefCharElem`. See §15.4.2 [Treating FrameMaker format names as DITA element names](#) on page 486.

15.4.4.4 Overriding element mapping for character formats

If mapping a `FrameMaker` character format does not suffice for a phrase element, you can use **DITASStartElem** and **DITAEEndElem** markers placed at the start and end, respectively, of the character span to be delimited as a phrase element. The content of each marker is the tag name for the inline element; **Mif2Go** provides the `< >` and `</ >`. You cannot use a **DITATag** marker to override the element-name mapping for an inline element.

15.4.4.5 Using alternate character formats for menu cascades

In `FrameMaker` it is not possible to distinguish between two separate applications of the same character format to adjacent text spans, and one application of the format to both spans. Because the DITA `<menucascade>` element allows only `<uicontrol>` as content, text is excluded; you cannot use spaces, or any other character outside the `<uicontrol>` format, as separators.

The workaround is to create two character formats; for example, *mc1* and *mc2*; and apply them alternately to `<uicontrol>` elements when those elements are in a `<menucascade>`. You would map both formats to `<uicontrol>`:

```
[DITACCharTags]
mc* = uicontrol
```

And indicate that the elements mapped from both formats must be in a `<menucascade>`:

```
[DITAParents]
mc* = menucascade
```

See §15.5.2 [Designating DITA ancestor elements](#) on page 502.

15.4.5 Assigning multiple typographic elements to a format

Mif2Go suppresses overrides that are not part of a paragraph or character format, and uses only the single element mapped from the format name in `[DITAParaTags]` or in `[DITACCharTags]`. This can be problematic if, for example, a character format should map to both `` and `<i>`. In that case, you have to map the format to one of the elements in `[DITACCharTags]`, and assign the other(s) as follows:

```
[DITACCharTypographics]
; Frame character format (wildcards OK) = DITA typographic elements
CharFmtName = typelem1 typelem2 ...
```

Likewise for paragraph formats:

```
[DITAParaTypographics]
; Frame paragraph format (wildcards OK) = DITA typographic elements
ParaFmtName = typelem1 typelem2 ...
```

You can assign any or all of `b`, `i`, `u`, `tt`, `sup`, or `sub`, in addition to the element to which the format is mapped in `[DITAParaTags]` or in `[DITACHarTags]`. You *must* map the format to an element (not to `No`) in `[DITAParaTags]` or in `[DITACHarTags]`, then add the rest of the elements (but *not* the one already mapped) here. For example:

```
[DITACHarTags]
CodeBoldItal = tt
```

List the elements separated by spaces:

```
[DITACHarTypographics]
CodeBoldItal = b i
```

The tags are applied in the order listed. For example, with these settings a *CodeBoldItal* text fragment would be enclosed in `<tt><i>...</i></tt>`.

We advise minimal use of this feature. Typographic presentation markup is best left to later processing, such as with a CSS rule based on the `outputclass` attribute of the DITA semantic element. For example, map character format *CodeBoldItal* to `<ph>`, and expect the HTML output to produce ``, which can be handled by CSS for selector `span.codeboldital`.

See §15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498.

15.4.6 Assigning attributes to DITA elements

You can apply attributes to a DITA block or inline element by assigning *attribute=* "value" pairs to the FrameMaker format mapped to the element. The attributes you assign with configuration settings apply to all instances of the element in question. Only those attributes assigned to elements mapped from paragraph formats can be overridden with markers.

The following are special cases:

- A value for the `id` attribute can be assigned only with a **DITAElemID** marker.
- Attributes of `<xref>` elements require different settings; see §15.10 [Configuring cross references and links for DITA](#) on page 527.
- The `outputclass` attribute can be assigned to the root element only with a **DITATopicOutputclass** marker.

In this section:

§15.4.6.1 [Specifying a value for the id attribute](#) on page 495

§15.4.6.2 [Including an id attribute in every element](#) on page 496

§15.4.6.3 [Specifying attribute values for the root element of a topic](#) on page 497

§15.4.6.4 [Specifying attribute values for a block element or parent](#) on page 497

§15.4.6.5 [Specifying attribute values for an inline element](#) on page 498

§15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498

See also:

§15.6 [Converting tables to DITA XML](#) on page 510

15.4.6.1 Specifying a value for the id attribute

To specify an ID for a block element, place a **DITAElemID** marker in the FrameMaker paragraph. The content of the marker is the value of the `id` attribute. When you place a **DITAElemID** marker at the start of a topic, the content of the marker becomes the `id` attribute of the `<title>` element for that topic.

Note: For an embedded topic (as opposed to a block element), you must use a **DITATopicID** marker instead; see §15.9.3 [Specifying the ID for a DITA topic](#) on page 526.

If a paragraph does not contain a **DITAElemID** marker, **Mif2Go** uses the content of the first **newlink** marker in the paragraph as the ID for the element. If there is no **newlink** marker, **Mif2Go** uses for the ID a combination of the **Mif2Go** FileID of the FrameMaker file and the FrameMaker ObjectID of the paragraph; see §5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117. You can override the **Mif2Go**-assigned ID with a **DITAElemID** marker.

Note: The `id` attribute value must start with a letter. All **Mif2Go**-assigned FileIDs conform to this requirement. If you want your DITA XML output to validate, avoid changing any FileID values in `mif2go.ini` to start with a digit.

Mif2Go provides a default `id` attribute for each of the following block elements:

- `<table>` (all types)
- `<fig>` (but not `<image>`)
- `<section>`
- `<example>`
- `<refsyn>`
- `<fn>`
- ``
- `<p>` (when the paragraph contains index markers or starts a FrameMaker page).

Links to any of these elements automatically pick up the `id` attribute, and also the correct `type` attribute of the element. For links to other elements, you have to insert either a **newlink** marker or a **DITAElemID** marker in the target paragraph in FrameMaker, and specify the link `type` attribute with a **DITALinkType** marker; see §15.10.6.3 [Specifying the `<xref>` type attribute](#) on page 530.

*Interpolated
parent id attribute*

When the parent of the current block element is interpolated by **Mif2Go** (see §15.4.1 [Understanding how Mif2Go delimits DITA elements](#) on page 486), you cannot use a **DITAElemID** marker to specify an ID for that parent.

To specify an ID for the parent of the current block element, place a **DITAParentID** marker in the element, with content as follows:

```
parentname=parentid
```

Do not include spaces around the equals sign.

15.4.6.2 Including an id attribute in every element

By default, **Mif2Go** automatically provides `id` attributes only for elements that require them; basically, any element that is a link target. To direct **Mif2Go** to include an `id` attribute in every element where the `id` attribute is valid:

```
[DITAOptions]
; SetElementIDs = No (default) or Yes (add @ids wherever possible)
SetElementIDs = Yes
```

When `SetElementIDs=Yes`, **Mif2Go** constructs an ID to use for the `id` attribute of every element for which an `id` attribute is valid, provided the element does not already have an `id` attribute assigned some other way, such as with a **DITAElemID** marker (see §15.4.6.1 [Specifying a value for the id attribute](#) on page 495).

See also:

§5.3 [Identifying files and objects](#) on page 117

15.4.6.3 Specifying attribute values for the root element of a topic

To apply attributes to the root element of the current topic, assign *attribute="value"* pairs, separated by spaces, to the FrameMaker paragraph format for the topic title:

```
[DITATopicRootAttrs]
; Frame para format for topic title (wildcards OK) = attributes for
; the root element of the current topic.
ParaFmt = attribute1="value1" attribute2="value2" ...
```

For example, for attributes to support another tool such as Docato that you will use to manage the DITA XML output, suppose your FrameMaker paragraph format for concept topic titles is *ConHead*:

```
[DITATopicRootAttrs]
ConHead = xmlns:xsa3="http://dita.oasis-open.org/architecture/2005/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../catalog/concept.xsd"
```

Of course this assignment would have to be all on one line in your configuration file, though it might not look that way here.

You can override root-element attributes with a **DITATopicRootAttr** marker.

15.4.6.4 Specifying attribute values for a block element or parent

For attributes of block elements, you can do the following:

[Assign block element attributes](#)

[Override block element attributes](#)

[Assign interpolated parent attributes](#)

[Override interpolated parent attributes](#)

When you want to override default or assigned attributes, keep in mind:

[Where to use DITAAttribute markers](#)

*Assign block
element attributes*

To apply attributes (other than *id*) to a block element (other than *<xref>*), assign *attribute="value"* pairs, separated by spaces, to the paragraph format(s) mapped to the element:

```
[DITAParaAttributes]
; Frame para format (wildcards OK) = attributes
ParaFmt = attribute1="value1" attribute2="value2" ...
```

You can use **Mif2Go** macros for any part of the assignment, or even for the entire assignment. For example:

```
[DITAParaAttributes]
ParaFmt = <$MacroToWriteAttrs>
```

*Override block
element attributes*

To override a setting in `[DITAParaAttributes]` or to override default attributes for a particular instance of a block element, place a **DITAAttribute** marker in a paragraph mapped to the element, with content as follows:

```
elementname: attribute1="value1" attribute2="value2" ...
```

For example:

```
linklist: role="friend" type="reference"
```

The name of the element must be followed by a colon. Separate *attribute="value"* pairs with a space. Each value must be enclosed in double quotes. You can use **Mif2Go** macros for everything after the colon.

*Assign
interpolated
parent attributes*

To assign attributes to an interpolated parent of a block element:

```
[DITAParentAttributes]
; Frame para format (wildcards OK) = parentname: attributes
ParaFmt = parentname: attribute1="value1" attribute2="value2" ...
```

You can use **Mif2Go** macros for the assignment.

*Override
interpolated
parent attributes*

To override a setting in [DITAParentAttributes] or to override default attributes for an interpolated parent of a block element, place a **DITAAttribute** marker in a paragraph mapped to the element, with content as follows:

```
parentname: attribute1="value1" attribute2="value2" ...
```

To apply attributes to more than one interpolated parent, use a separate marker for each parent.

*Where to use
DITAAttribute
markers*

Use **DITAAttribute** markers only to supply attribute values other than the DTD default values for an element, or to override attribute values specified in a configuration file. Do not use **DITAAttribute** markers for any of the following:

- The `id` attribute of the current element; use a **DITAElemID** marker instead. See §15.4.6.1 [Specifying a value for the id attribute](#) on page 495.
- The `id` attribute of an interpolated parent of the current element; use a **DITAParentID** marker instead. See §15.4.6.1 [Specifying a value for the id attribute](#) on page 495.
- Any `<xref>` element attributes; use **DITALink*** markers instead. See §15.10.6 [Overriding <xref> attribute values](#) on page 529.

A **DITAAttribute** marker overrides settings in [DITAParaAttributes] and [DITAParentAttributes], but does not override settings in [DITACHarAttributes] (see §15.4.6.5 [Specifying attribute values for an inline element](#) on page 498) or [DITATableAttributes] (see §15.6 [Converting tables to DITA XML](#) on page 510).

15.4.6.5 Specifying attribute values for an inline element

To apply attributes (other than `id`) to an inline element, assign `attribute="value"` pairs, separated by spaces, to the character format(s) mapped to the element:

```
[DITACHarAttributes]
; Frame char format (wildcards OK) = attributes
CharFmt = attribute1="value1" attribute2="value2" ...
```

You cannot use markers to override settings in [DITACHarAttributes].

15.4.6.6 Providing outputclass attributes for all elements

By default, **Mif2Go** does not try to assign CSS classes to DITA elements (unless you set [CSS]UseCSS=Yes; see §22.4 [Specifying CSS file and link options](#) on page 683).

To direct **Mif2Go** to provide an `outputclass` attribute for elements that allow this attribute:

```
[DITAOptions]
; UseOutputClass = No (default when [CSS]UseCSS=No)
; or Yes (default when [CSS]UseCSS=Yes)
UseOutputClass = Yes
```

When `UseOutputClass=Yes`, **Mif2Go** includes an `outputclass` attribute where allowed. The value of the attribute is the [ParaClasses] or [CharClasses] assignment (if any) for the FrameMaker format mapped to the element, otherwise the name of the FrameMaker format mapped to the element. See §22.7 [Assigning CSS classes](#) on page 691.

Note: To include `outputclass` attributes, make sure your configuration file does *not* specify `[CSS]WriteClassAttributes=No`.

Root element outputclass The only way to include an `outputclass` attribute in the root element is with a **DITATopicOutputclass** marker: place the marker anywhere in the topic, and make the content of the marker the value of the `outputclass` attribute.

15.4.6.7 Converting FrameMaker change bars to condition attributes

When `UseTypographicElements=Yes`, **Mif2Go** creates DITA `<chbar>` elements from text that has the FrameMaker character property *change bar*; see §15.4.4.2 [Including typographic elements in addition to mapped formats](#) on page 493.

If instead you want to apply an attribute value to this text (such as `@status="changed"`), you can use the following workaround:

1. Create a condition in FrameMaker named (for example) *Changed*.
2. In FrameMaker, find all text that has the character format property *change bar* and change by pasting the conditional text setting *Changed*. (This could be done with FrameScript.)
3. In your project configuration file, include the following setting:

```
[ConditionAttributes]
Changed = status="changed"
```

If an entire paragraph is conditional (including the end-of-paragraph pilcrow), the resulting element will have `<p status="changed">`. If only part of a paragraph is conditional, the element will have `<ph status="changed">`. See §15.12.2 [Mapping FrameMaker conditions to element attributes](#) on page 533.

15.4.7 Preserving whitespace in block elements

To make sure content will be processed with whitespace preserved as it is in your FrameMaker document:

```
[DITAPreformatted]
; element name = Yes (default for <codeblock>)
; or No (default for all other block elements)
codeblock = Yes
```

When `elementname=Yes`, **Mif2Go** processes the element in question with whitespace unchanged from that in FrameMaker, as in HTML `<pre>` elements (see §21.10 [Configuring preformatted text for HTML/XML](#) on page 670).

15.4.8 Including Pls for line, column, or page breaks

To force a line break, column break, or a page break when non-preformatted text in DITA is rendered, you can use a **Code** marker or a macro to insert one (or more) of the following processing instructions in FrameMaker at the desired break point:

<code><?dthtm Break="line" ?></code>	HTML line break
<code><?dtrtf Break="line" ?></code>	RTF line break
<code><?dtrtf Break="column" ?></code>	RTF column break
<code><?dtrtf Break="page" ?></code>	RTF page break

For example, to force an HTML-only line break at a point where you have a forced return (**Shift+Enter**) in FrameMaker, immediately after the forced return insert a **Code** marker with the following content:

```
<?dthtml Break="line" ?>
```

To include a processing instruction for every forced return in a given paragraph format, you can define a macro to find the forced returns and insert the processing instructions; see §28.6 [Using expressions in macros](#) on page 811.

15.4.9 Providing a <shortdesc> element for a DITA topic

If you use a dedicated paragraph format for topic abstracts in your FrameMaker document, you can simply map that format to `shortdesc`; see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487. The content of any paragraph in that format becomes the `<shortdesc>` element for the enclosing topic.

To add a `<shortdesc>` element to a topic without using a paragraph format, place a **DITAShortDesc** marker in the topic. The content of the **DITAShortDesc** marker is the text of the abstract.

Whichever way you provide the content, the resulting `<shortdesc>` element appears in the map, in the `<topicmeta>` of the `<topicref>` for the topic.

15.4.10 Converting index markers to <indexterm> elements

Mif2Go automatically converts the content of FrameMaker **Index** markers to DITA `<indexterm>` elements, determining index levels according to the conventions and settings described in §7.5 [Configuring index entries for Help systems](#) on page 211.

A multiple-level index entry in FrameMaker becomes a set of nested `<indexterm>` elements. For example, an index entry with the following content:

```
adaptive table sizing:for HTML:overriding
```

becomes:

```
<indexterm>adaptive table sizing
  <indexterm>for HTML
    <indexterm>overriding
  </indexterm>
</indexterm>
```

When `DITAVer=1` (see §15.2.3 [Specifying DITA output options](#) on page 480), **Mif2Go** produces elements `index-see`, `index-see-also`, and `index-sort-as`, and converts `<$startrange>` and `<$endrange>` **Index** markers to include DITA index range attributes. For example:

```
<$startrange>converting:characters
```

becomes:

```
<indexterm>converting
  <indexterm start="converting: characters">characters</indexterm>
</indexterm>
```

According to the DITA specification, `<indexterm>` is invalid in a `<title>` element. Therefore, if an index marker is in a FrameMaker paragraph mapped to the DITA `<title>` element, **Mif2Go** wraps the `<indexterm>` element in a `<ph>` element.

15.5 Nesting DITA block elements

Nesting block elements is the most challenging aspect of treating unstructured FrameMaker as though it were structured. This section discusses ancestry of paragraph-based block elements.

In this section:

- §15.5.1 [Understanding how Mif2Go determines element nesting](#) on page 501
- §15.5.2 [Designating DITA ancestor elements](#) on page 502
- §15.5.3 [Fixing up interpolated ancestries](#) on page 503
- §15.5.4 [Deciding when to fully specify ancestry](#) on page 503
- §15.5.5 [Specifying alternate ancestries for the same element](#) on page 504
- §15.5.6 [Avoiding invalid ancestries](#) on page 504
- §15.5.6 [Avoiding invalid ancestries](#) on page 504
- §15.5.8 [Configuring nested lists](#) on page 505
- §15.5.9 [Closing DITA ancestor elements](#) on page 506
- §15.5.10 [Opening DITA ancestor elements](#) on page 507
- §15.5.11 [Configuring multi-paragraph list items](#) on page 508
- §15.5.12 [Splitting a paragraph into separate DITA elements](#) on page 508
- §15.5.13 [Specifying DITA element levels](#) on page 509

See also:

- §15.6.3 [Designating ancestors for <table> elements](#) on page 512
- §15.7.1 [Designating ancestors for <image> and <fig> elements](#) on page 516

15.5.1 Understanding how Mif2Go determines element nesting

For each element, **Mif2Go** considers whether that element can go inside the current parent element. If not, **Mif2Go** uses heuristic methods based on the possible parents, level limitations, and current context. To provide a parent element **Mif2Go** selects, in alphabetical order, the first wrapper element that can validly fit and that is permitted by your settings. *Mif2Go does not interpret the DTD to determine element nesting.*

For example, in the absence of project configuration settings that designate valid parent elements, you might find that most of your content ends up nested in <abstract>, because “abstract” is near the beginning of the alphabet.

As another example, suppose your document uses a sequential structure for steps in a procedure: paragraph format *Step1* for the first step, followed by several *StepNext* paragraphs, all containing both commands and informational text. To convert this structure to a hierarchical DITA structure, with paragraphs in both formats becoming <step> children of a <steps> element, you would specify just one setting (see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487):

```
[DITAParaTags]
Step* = step
```

The first paragraph in the group forces creation of <steps>, because DITA requires <steps> or <steps-unordered> as the parent of <step>, and of the two valid candidate parents, <steps> comes first alphabetically. As soon as **Mif2Go** encounters a paragraph format mapped to an element that is not valid in <steps>, the parent tag is closed.

For problem cases, you can use a **DITALevel** marker to explicitly set the level for an element; see §15.5.13 [Specifying DITA element levels](#) on page 509. However, for nested lists, use a different approach; see §15.5.5 [Specifying alternate ancestries for the same element](#) on page 504.

15.5.2 Designating DITA ancestor elements

For block elements such as `` that can have more than one possible ancestry, map any paragraph formats to the intended (required) parent element, and if necessary, grandparent element, even great-grandparent element. List ancestors in hierarchical order. Specify a parent only if it is required to prevent incorrect output. If you find “Parent Error” comments in the resulting XML, first try commenting out any related parent assignment in `[DITAParents]`.

Note: Do not specify parents for paragraph formats that are part of a FrameMaker table, nor for the table itself (see §15.6.3 [Designating ancestors for <table> elements](#) on page 512). If necessary you can specify parents for the contents of table cells, but do not go above `<entry>`. See §15.6.5.1 [Omitting ancestries of DITA table components](#) on page 513.

To specify required ancestors of elements mapped from FrameMaker formats (for example):

```
[DITAParents]
; Frame para format (wildcards OK) = required parents
Title = topic
Heading* = section
Numbered1 = ol li
Numbered = ol li
Bulleted = ul li
Figure Title = fig
Syntax = refsyn
Example = example
```

In this example a *Numbered1* paragraph (which is mapped to `<p>` in `[DITAParaTags]`) must have these ancestors:

```
<ol>...<li>...</li>...</ol>
```

Therefore, each *Numbered1* paragraph starts a new `` if and only if an `` is not already open; and starts a new `` if and only if an `` under the `` is not already open. To force a new `` for *Numbered1* paragraphs, you must also give the *Numbered1* paragraph format first-child status under both parent and grandparent elements; see §15.5.6 [Avoiding invalid ancestries](#) on page 504.

Note: For list items that can include more than one paragraph, map the paragraph format to `<p>`, then designate its including list element as a parent.

Use this mapping for formats such as lists, in which `` elements are needed under `` or `` in addition to the `<p>` elements mapped in `[DITAParaTags]`.

List ancestors in hierarchical order

If a parent element has more than one possible parent, and only one of those parents can be a grandparent of the paragraph format in question, list both the grandparent and parent, in hierarchical order.

Override individual ancestries

To override the `[DITAParents]` assignment for a given instance of a paragraph format, place a **DITAParent** marker in the paragraph. Make the content of the marker the name(s) of the ancestor element(s), in hierarchical order. A **DITAParent** marker specifies the required ancestry for the current block element, overriding whatever is specified in `[DITAParents]`.

Not for
FrameMaker
table elements

Although you can specify ancestors for tables (see §15.6.3 [Designating ancestors for <table> elements](#) on page 512), do not specify parents for paragraph formats that are part of a FrameMaker table. **Mif2Go** uses a different mechanism for table components. You can specify parents for the contents of table cells, but do not go above <entry>. See §15.6.5.1 [Omitting ancestries of DITA table components](#) on page 513.

See also:

§15.6.3 [Designating ancestors for <table> elements](#) on page 512

§15.7.1 [Designating ancestors for <image> and <fig> elements](#) on page 516

15.5.3 Fixing up interpolated ancestries

Creating DITA structure from FrameMaker formats necessarily involves some trial and error. When you see unexpected interpolation of inappropriate parent elements in your output, it is usually because you have not specified parents for a particular format-to-element mapping. For example, suppose you map paragraph format *Ref* to <p>, and use a *Ref* paragraph at the top level of each reference topic, where <p> is not valid. On encountering a *Ref* paragraph in this situation, with no parents specified for the *Ref* format, **Mif2Go** would go through the list of valid parents for <p> in a reference topic, and interpolate the first set that works; which might be <codeblock><draft-comment>.

The remedy is to figure out what would be a more appropriate lineage for the element in question. You could specify that lineage for the format in [DITAParents] if it applies generally, or insert a **DITAParent** marker in the paragraph for an isolated instance. In this example, the following mapping would produce better results:

```
[DITAParents]
Ref = refbody section
```

The **Mif2Go** search algorithm finds the shortest path, but that is not always the only shortest path, or the best path.

See also:

§15.4.3.4 [Omitting invalid tags for default DITA block elements](#) on page 489

§15.5.6 [Avoiding invalid ancestries](#) on page 504

15.5.4 Deciding when to fully specify ancestry

You do not need to map paragraphs in [DITAParents] for elements that can have only one possible ancestry, or for cases where **Mif2Go** can determine heuristically which of the possible ancestors fits the context best. Specify ancestry in [DITAParents] when more than one lineage is possible in the context of use.

Include as many ancestors as necessary to fully specify ancestry for the element to which a paragraph format is mapped in [DITAParaTags]. If your document includes actual instances of different ancestries for the same element, use sets of ancestors to specify the alternatives; see §15.5.5 [Specifying alternate ancestries for the same element](#) on page 504. In some cases you might have to include all ancestors up to the topic level, and you might have to determine this necessity by trial and error; that is, list them all whenever *not* including all ancestors causes unwanted nesting.

When **Mif2Go** encounters a set of ancestors specified either in [DITAParents] or in a **DITAParent** marker, **Mif2Go** tries to nest the ancestor hierarchy in the current element. If the entire hierarchy is valid in that position, that is where it stays. This means that if your source document uses paragraph format *Body* (for example) for all text that is not nested in a list, and you map *Body* to DITA element <p>, you must also specify non-list parents for *Body*, because <p> can nest in ; in fact, in almost any block element. Unless you can

make sure every block element that could precede a *Body* paragraph gets closed (see §15.5.9 [Closing DITA ancestor elements](#) on page 506), the *Body* <p> is likely to be nested in the preceding element.

15.5.5 Specifying alternate ancestries for the same element

If your document uses the same paragraph format in several lineages, you can create a set of alternate ancestor elements for **Mif2Go** to choose from, depending on the context. The following predefined element sets are included in your project configuration file when you first set up a DITA project. You can alter or delete these sets, and you can define additional sets.

To define sets of elements to be considered as alternate ancestors:

```
[DITAElementSets]
; $setname = DITA elements in the set.
; These element sets are predefined in the starting .ini for DITA:
$top = body conbody taskbody refbody section step
$text = body conbody taskbody refsyn section step entry stentry
$list= ol ul
```

Each set name must start with a dollar sign (\$). You must define each set as a collection of elements; you may *not* define one element set in terms of other element sets.

You can use an element set name in place of an element name in [DITAParents], in [DITAFirst], or in the corresponding **DITAParent** and **DITAFirst** markers. For example:

```
[DITAParents]
Body = $text
Body2 = $top $list li
```

Any element in the set is acceptable at the point where it appears in the hierarchical sequence. There is no equivalent marker.

See also:

§15.5.6 [Avoiding invalid ancestries](#) on page 504

15.5.6 Avoiding invalid ancestries

For the purpose of constructing ancestries, by default **Mif2Go** treats *topic* as a synonym for *concept*, *task*, *reference*, *glossentry*, and any other topic type, and treats *body* as a synonym for any of the body types, such as *conbody*. This can cause invalid interpolated ancestries, because **Mif2Go** might include an element, or wrap an element in a parent, that is not valid for the topic type.

To avoid this problem and direct **Mif2Go** to treat *topic* and *body* as applying only to the generic topic type:

```
[DITAOptions]
; UseCommonNames = Yes (default, in [DITAParents] and
; in [DITAElementSets], treat "topic" as a synonym for
; concept, task, reference, glossentry, and any other
; topic type, and treat "body" as a synonym for any of
; the body types like conbody), or No (treat topic and
; body as applying only to the generic "topic" type)
UseCommonNames = No
```

For example, when *UseCommonNames*=Yes (the default) and *UseFormatAsTag*=Yes (see §15.4.2 [Treating FrameMaker format names as DITA element names](#) on page 486), a paragraph whose format is *Body* will be allowed as a <body> element in a reference topic, where <body> is not valid.

See also:

§15.5.5 [Specifying alternate ancestries for the same element](#) on page 504

15.5.7 Specifying first-child status for nested elements

To specify parent elements in which the paragraph format mapped to a given block element must appear as the first child:

```
[DITAFirst]
; Frame para format = parents under which the current block element
; (or one of its parents) must be the first child.
Numbered1 = ol li
Numbered = li
Bulleted = li
```

If the parent element you assign to a paragraph format has more than one possible parent, and the paragraph format in question needs to be first only for one of its possible grandparents, list both the grandparent and parent, separated by spaces. You can list as many ancestors as necessary to fully specify first-child status for the paragraph format. List the ancestors in hierarchical order. The list must match the ancestor list in [DITAParents]; see §15.5.2 [Designating DITA ancestor elements](#) on page 502.

Use these settings mainly for lists, to ensure that a paragraph format starts a new list item or a new list. For example, these settings specify the following for the list paragraph formats mapped to <p> in [DITAParaTags]:

- A *Numbered1* <p> element must be the first child of its parent element, which element must be the first child of its parent; this setting forces first-child status for the entire lineage of the elements listed, not just the last.
- A *Numbered* <p> element or a *Bulleted* <p> element must be the first child of its parent element.

If you use definition lists or parameter lists, you must specify first-child status for the paragraph format of the term. For example:

```
[DITAFirst]
DefTerm = dlenry
ParamTerm = plentry
```

To override the [DITAFirst] assignment for a given instance of a paragraph, place a **DITAFirst** marker in the paragraph. Make the content of the marker the name(s) of the desired ancestor element(s), in hierarchical order. A **DITAFirst** marker specifies that the current block element must be the first child of its listed ancestor elements, overriding whatever is specified in [DITAFirst].

15.5.8 Configuring nested lists

If your document includes nested ordered or unordered lists (or a mix of the two), it is best to specify a set of ancestor elements that includes both; see §15.5.5 [Specifying alternate ancestries for the same element](#) on page 504. For example:

```
[DITAElementSets]
$list= ol ul
```

This particular element set is predefined in the starting configuration file for DITA output. Specifying \$list as an ancestor lets you have bullets subordinate to either bulleted or numbered items, and vice-versa:

In your list items you must use <p>, not just , because the nested needs to be inside the , and the smallest enclosing tag always closes at the end of the

paragraph (to prevent “pernicious mixed content” wherever possible). This way, the <p> closes, but the stays open for the nested list.

Suppose your document includes a hierarchy of paragraph formats like this:

```

Body
  Bulleted
    BulletedSub
    BulletedSub
  Bulleted
    BulletedSub
    BulletedSub
    BulletedSub
Body

```

You would specify the following settings for the bulleted items:

```

[DITAParaTags]
Bulleted = p
BulletedSub = p

[DITAParents]
Bulleted = $text ul li
BulletedSub = $text $list li ul li

[DITAFirst]
Bulleted = li
BulletedSub = li

```

Note: Do *not* try to use DITA levels to nest lists.

15.5.9 Closing DITA ancestor elements

To get a block element under the correct parent, you might have to specify that an ancestor element (and all its descendants) must end when the current block element ends; or that the prior block must end before the current block element begins.

In this section:

§15.5.9.1 [Ending ancestor elements before the current block](#) on page 506

§15.5.9.2 [Ending ancestor elements after the current block](#) on page 507

15.5.9.1 Ending ancestor elements before the current block

In some cases, it is not clear whether a paragraph is supposed to be a child of the preceding element (or nest of elements). For example, by default a <p> element following a list item becomes part of the , and that is not necessarily what you want.

To close an element (or a hierarchy of elements) before starting the current block (for example):

```

[DITACloseBefore]
; Frame para format = elements to be closed, with any other elements
;   nested under them, before the current block element starts.
Recap = li
Body = ul ol

```

Use this setting to force closure of elements that were opened based on settings in [DITAParents]; see §15.5.2 [Designating DITA ancestor elements](#) on page 502. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DITACloseBefore** marker in the paragraph for the current block element instead, with content the name(s) of the element(s) to close. You can also use a **DITACloseBefore** marker to override a [DITACloseBefore] setting when you want to close a higher (or lower) ancestor than the setting specifies.

15.5.9.2 Ending ancestor elements after the current block

In some cases, it is not clear whether the end of a block element should also end the enclosing parent element. For example, if your document includes illustrations with the figure title above the image, body paragraphs following the image would normally be included in the <fig> element, because the content model allows <p> inside <fig>.

To close a parent element at the end of the current block element (for example):

```
[DITACloseAfter]
; Frame para format = parent to be closed, with any other elements
; nested under it, at the end of the current block element.
FigAnchor = fig
```

Use this setting to force closure of elements that were opened based on settings in [DITAParents]; see §15.5.2 [Designating DITA ancestor elements](#) on page 502. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DITACloseAfter** marker in the paragraph for the current block element instead, with content the name(s) of the ancestor element(s) to close. You can also use a **DITACloseAfter** marker to override a [DITACloseAfter] setting when you want to close a higher (or lower) ancestor than the setting specifies.

15.5.10 Opening DITA ancestor elements

To get a block element in the correct position in a hierarchy, you might have to force the opening of interpolated ancestor elements first; or, in some cases, specify elements that must be opened after the current element ends.

In this section:

§15.5.10.1 [Starting ancestor elements before the current block](#) on page 507

§15.5.10.2 [Starting a new hierarchy after the current block](#) on page 507

15.5.10.1 Starting ancestor elements before the current block

To open interpolated ancestor elements before starting the current block:

```
[DITAOpenBefore]
; Frame para format = elements to be opened, with any other elements
; nested under them, before the current block element starts.
somefmt = someancestor
```

Use this setting to force opening of elements when [DITAParents] does not suffice.

For individual cases, you can insert a **DITAOpenBefore** marker in the paragraph for the current block element instead, with content the name(s) of the element(s) to open. You can also use a **DITAOpenBefore** marker to override a [DITAOpenBefore] setting when you want to open a higher (or lower) ancestor than the setting specifies.

15.5.10.2 Starting a new hierarchy after the current block

To open a new element or hierarchy of elements after the current block ends:

```
[DITAOpenAfter]
; Frame para format = elements to be opened, with any other elements
; nested under them, after the current block element ends.
somefmt = someancestor
```

Use this setting to force opening of elements when [DITAParents] does not suffice.

For individual cases, you can insert a **DITAOpenAfter** marker in the paragraph for the current block element instead, with content the name(s) of the element(s) to open. You can

also use a **DITAOpenAfter** marker to override a [DITAOpenAfter] setting when you want to open an element or hierarchy other than what the setting specifies.

15.5.11 Configuring multi-paragraph list items

By default, at the end of each paragraph **Mif2Go** closes the block element to which the paragraph format is mapped (see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487). If any list items in your document include multiple paragraphs or sublists, you must make sure that each can include more than one block element, but also that the last item in each list or sublist does not slurp up any following paragraphs.

To configure list elements:

Map formats to <p> instead of .

Specify ancestry for each format.

Make each format first in .

Make sure each list ends where it should.

*Map formats to
<p> instead of
*

Map all list-item paragraph formats to <p> rather than to ; for example:

```
[DITAParaTags]
Numbered1 = p
Numbered = p
Bulleted = p
BulletedLast = p
```

*Specify ancestry
for each format*

Designate the appropriate ancestors for each type of list element:

```
[DITAParents]
Numbered1 = ol li
Numbered = ol li
Bulleted = ul li
BulletedLast = ul li
```

*Make each format
first in *

Make sure each list-item paragraph is first in its element:

```
[DITAFirst]
Numbered1 = ol li
Numbered = li
Bulleted = li
BulletedLast = li
```

*Make sure each
list ends where it
should*

If the last paragraph in a multi-paragraph list item is followed by a paragraph whose format is mapped to an element that is valid in , that paragraph will be included in the list item. To prevent including the following paragraph, you can explicitly close the list:

```
[DITACloseAfter]
BulletedLast = ul li
```

Or insert a **DITACloseAfter** marker in the last list-item paragraph, with content `ul li`.

As an alternative, you can make sure the list closes before the following paragraph:

```
[DITACloseBefore]
Body = ul ol
```

Or insert a **DITACloseBefore** marker in the following paragraph, with content `ul ol`.

15.5.12 Splitting a paragraph into separate DITA elements

Suppose your FrameMaker document uses a paragraph format named *Definition*. The content of a *Definition* paragraph consists of a term, followed by a tab character, followed by the definition of the term. And suppose you want to convert these paragraphs to a DITA definition list, with each *Definition* paragraph divided as follows:

```
<dlentry>
  <dt>term</dt>
  <dd>definition</dd>
</dlentry>
```

To specify DITA settings for the paragraph format:

```
[DITAParaTags]
; Each paragraph starts with a term:
Definition = dt

[DITAFirst]
; Each paragraph becomes a separate dlentry:
Definition = dlentry
```

The second setting above is required because a DITA `<dlentry>` element can start with more than one `<dt>` element; however, in this case, you would want each `<dt>` to begin a new `<dlentry>`.

Breaking each paragraph at the tab character requires a **Mif2Go** macro. The macro requires capturing the content of each *Definition* paragraph for parsing:

```
[HTMLParaStyles]
Definition = CodeStore CodeAfter
```

Mif2Go first surrounds the content of each *Definition* paragraph with tags as specified in `[DITAParaTags]`, then stores the content in `CodeStore` variable `$$Definition`; see §28.3.7.2 [Inserting code with the CodeStore property](#) on page 804.

The `CodeAfter` property takes care of placing the result of macro expansion in the output; see §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822:

```
[ParaStyleCodeAfter]
Definition = <$DefMacro>
```

The macro must check the content for a tab character. However, tabs are converted to spaces, because they are not meaningful in XML, so a compare to a tab would always fail; see §21.6.2 [Understanding how Mif2Go treats tabs in HTML/XML](#) on page 658. Instead, the macro compares to a space:

```
[DefMacro]
; $$Definition contains "<dt>term def ... </dt>"
<$term = ($$Definition before " ")>\
<$defn = (($$Definition after " ") before "</dt>")>\
<$term></dt>
<dd><$defn></dd>
```

This macro works correctly only when there are no terms that contain spaces. If some terms contain spaces, you would have to devise a different method, perhaps applying a character format to each term in FrameMaker.

15.5.13 Specifying DITA element levels

To specify the level at which a block element should appear in DITA output, you can assign a level number to any FrameMaker paragraph formats that are mapped to the element (see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487). However, for most nesting issues, you should use settings that specify ancestry rather than level; see §15.5.2 [Designating DITA ancestor elements](#) on page 502.

Assign levels only for the following purposes:

- to identify paragraph formats mapped to `<title>` that should start new topics; assign level 1 to each such format
- to handle unusual situations that cannot be addressed any other way.

To specify the level of a DITA block element:

```
[DITALevels]
; Frame para format (wildcards OK) = level in DITA (not Frame) file
; required for the DITAParaTag specified for this element.
FmtName = N
```

The lower the level number, the higher the level; <topic> is level 0, the root. You cannot put anything else at level 0. The topic title is at level 1. The first heading in the topic body is at level 3 (a title below <topic>, <body>, and <section>).

Specify level 1 for each paragraph format that starts a topic. For example:

```
[DITALevels]
Title = 1
Heading* = 1
GlossItem = 1
```

Assign level 1 only to topic-title formats. If you assign level 1 to a paragraph format that does not start a topic, each topic in which such a paragraph occurs will end prematurely, and a new topic will start at the level-1 paragraph. Probably not what you want.

Do not try to use DITA levels to achieve nested lists; instead see §15.5.8 [Configuring nested lists](#) on page 505.

To override the assigned level of a particular paragraph, place a **DITALevel** marker in the paragraph. A **DITALevel** marker specifies the level at which the current block element should appear in the DITA file, overriding whatever is specified for the format in [DITALevels]. The content of a **DITALevel** marker is a single integer.

See also:

§16.2.2 [Specifying topic levels in ditamaps](#) on page 544

15.6 Converting tables to DITA XML

In this section:

§15.6.1 [Working with Mif2Go DITA table types](#) on page 510

§15.6.2 [Marking table footer rows for future reference](#) on page 511

§15.6.3 [Designating ancestors for <table> elements](#) on page 512

§15.6.4 [Applying attributes to DITA tables](#) on page 512

§15.6.5 [Configuring DITA table components](#) on page 513

§15.6.6 [Converting tables used only as image containers](#) on page 514

§15.6.7 [Omitting table coding entirely](#) on page 515

See also:

§32.7.7 [Providing table structure information for DITA topic types](#) on page 916

15.6.1 Working with Mif2Go DITA table types

To specify how the tables in your FrameMaker document should be treated in DITA XML, you must associate each table format with a DITA table construct. However, instead of mapping directly to DITA table element names, you must map the formats to **Mif2Go** table types. This is because more than one table format can refer to the same DITA element, but with different properties.

In this section:

§15.6.1.1 [Mapping table formats to Mif2Go table types](#) on page 511

§15.6.1.2 [Designating a default table type](#) on page 511

§15.6.1.3 [Assigning DITA attributes to table types](#) on page 511

15.6.1.1 Mapping table formats to Mif2Go table types

For exceptions, you can insert a **DitaTable** marker at the start of the table title, with content one of the established DITA types: simple, tgroup, properties, choice, or definition.

To map FrameMaker table formats to **Mif2Go** table types (for example):

```
[DITATables]
; Frame table format (wildcards OK) = base table type, one of table,
; simple, figure, choice (in task), property (in reference), or
; strip (omit table coding, keep title and cell content).
FormatA = table
Format B = simple
Choices = choice
Properties = property
FigTable = figure
Holder = strip
```

The names of table types assigned to table formats in [DITATables] are **Mif2Go** identifiers, not DITA table tags. You can define additional table types in content-model configuration files; see §32.7.7 [Providing table structure information for DITA topic types](#) on page 916.

15.6.1.2 Designating a default table type

To specify a default **Mif2Go** table type for tables not listed in [DITATables]:

```
[DITAOptions]
; DefTableElem = element for tables not listed in [DITATables],
; default is "table" which is the full, complex DITA table type.
DefTableElem = property
```

15.6.1.3 Assigning DITA attributes to table types

To assign attributes to a table type, list *attribute="value"* pairs separated by spaces:

```
[DITATableAttributes]
; table type = attributes to be used for every instance
tabletype = attribute1="value1" attribute2="value2" ...
```

You can use **Mif2Go** macros for any part of the attribute assignment. You cannot use markers to override settings in [DITATableAttributes].

15.6.2 Marking table footer rows for future reference

The DITA specification does not encompass the notion of a table footer row. However, you can assign an @outputclass value to the <row> and <strow> elements generated from your FrameMaker table footer rows, to maintain that information.

To assign an @outputclass to footer rows:

```
[DITAOptions]
; UseTableFooterClass = No (default, treat footer rows as
; body rows), or Yes (use body element but set outputclass)
UseTableFooterClass = Yes
; TableFooterClass = @outputclass to use for footer rows,
; default "footer".
TableFooterClass = footer
```

When `UseTableFooterClass=Yes`, each footer row of every FrameMaker table becomes a DITA `<row>` or `<strow>` element that includes an `@outputclass` attribute with the value assigned to `TableFooterClass`. The default is `@outputclass="footer"`.

When `UseTableFooterClass=No`, footer rows become table body rows in DITA output.

15.6.3 Designating ancestors for `<table>` elements

To specify the ancestor elements **Mif2Go** must use for `<table>` elements:

```
[DITAOptions]
; TableParents = parents for table tags, including simplettable
; and others; default none (use content model), may include sets
; from [DITAElementSets].
TableParents =
```

List ancestors in hierarchical order; see §15.5.2 [Designating DITA ancestor elements](#) on page 502. You can include element sets, as well as single elements; see §15.5.5 [Specifying alternate ancestries for the same element](#) on page 504. If you do not specify any ancestor elements, **Mif2Go** picks the first valid element listed in the content model, which might not be what you had in mind.

To specify ancestry for a single `<table>` element or a discrete group of `<table>` elements, assign the list to the table ID (see §24.2 [Defining sets of tables](#) on page 728). For example:

```
[TableGroup]
FormatA = chart
aa654321 = chart
FormatC = textframe
Unruled = textframe

[DITATableParents]
; table ID group (not type) = parents to be used for root table
element
chart = section
aa654321 = example
textframe = conbody
```

You can make a single `[DITATableParents]` setting in an `HTMConfig` marker, also; see §33.2.2 [Overriding settings with configuration markers](#) on page 921.

15.6.4 Applying attributes to DITA tables

Suppose you want to apply a DITA `outputclass` attribute to a table or group of tables.

If you want to be able to use `outputclass` for pretty much everything, based on the table format name (fixed for CSS use) for tables, and on either the FrameMaker paragraph or character format name or the `[Class]` setting for paragraph and character elements, just use `[DITAOptions]UseOutputClass=Yes`; see §15.4.6.6 [Providing outputclass attributes for all elements](#) on page 498.

To apply a DITA `outputclass` attribute value to table formats only:

```
[TableAttributes]
MySpecialTableFormatName = outputclass="myspecialclass"
```

Add any other special attributes you want for that table format on the same line, and use another line for each additional format; see §24.4.2 [Overriding attributes for selected tables](#) on page 736.

You can also specify an `outputclass` attribute for an individual table with a **TableOutputclass** marker in the text flow before the table anchor, or in the table title or first table row; see §24.1.2 [Understanding precedence of assignment methods](#) on page 728.

15.6.5 Configuring DITA table components

In this section:

- §15.6.5.1 [Omitting ancestries of DITA table components](#) on page 513
- §15.6.5.2 [Retaining empty paragraph tags in DITA table cells](#) on page 513
- §15.6.5.3 [Specifying relative vs. absolute widths for table columns](#) on page 513
- §15.6.5.4 [Including properties of table cells in DITA XML](#) on page 514
- §15.6.5.5 [Converting table titles to DITA XML](#) on page 514

15.6.5.1 Omitting ancestries of DITA table components

Do not specify required parents (see §15.5.2 [Designating DITA ancestor elements](#) on page 502) for elements that are part of a FrameMaker table. **Mif2Go** uses a different mechanism to determine nesting of table components. You *can* specify parents for the contents of table cells, but do not go above `<entry>`.

15.6.5.2 Retaining empty paragraph tags in DITA table cells

By default, for DITA output **Mif2Go** omits paragraph tags for otherwise empty non-preformatted paragraphs in table cells. However, you can choose to keep the tags:

```
[Tables]
; RemoveEmptyTableParagraphs = No (default)
; or Yes (DITA/DocBook default)
RemoveEmptyTableParagraphs = No
```

When `RemoveEmptyTableParagraphs=No`, paragraph tags (such as `<p></p>`) for empty paragraphs are retained in table cells in DITA XML.

When `RemoveEmptyTableParagraphs=Yes`, paragraph tags for empty paragraphs in table cells are omitted (except for preformatted text, where tags are always preserved). A table cell that is blank in FrameMaker (contains only empty paragraphs) would become just `<td></td>` in DITA XML output.

Note: This setting is independent of the setting for removing empty paragraphs in text; see §15.3 [Specifying general options for DITA](#) on page 483.

See also:

- §24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744

15.6.5.3 Specifying relative vs. absolute widths for table columns

DITA `<simpletable>` elements do not have absolute column widths or table widths; all you get are relative column widths. For valid DITA XML, you have to use a PI to set absolute column or table width. However, for `<table>` elements you can specify either absolute or relative column widths. By default, **Mif2Go** uses absolute widths.

To specify relative instead of absolute table and column widths for `<table>` elements:

```
[DITAOptions]
; TableColsRelative = No (default, in points using pt, in colspec
; width attributes) or Yes (in percents using *)
TableColsRelative = Yes
```

When `TableColsRelative=Yes`, for relative widths **Mif2Go** produces (for example):

```
<colspec colnum="1" colname="col1" colwidth="81*" />
<colspec colnum="2" colname="col2" colwidth="108*" />
```

Those relative widths (denoted by the `*`) turn inches into points, resulting in $1.125'' = 81\text{pt}$, and $1.5'' = 108\text{pt}$.

When `TableColsRelative=No`, **Mif2Go** produces instead:

```
<colspec colnum="1" colname="col1" colwidth="81pt" />
<colspec colnum="2" colname="col2" colwidth="108pt" />
```

15.6.5.4 Including properties of table cells in DITA XML

FrameMaker table cell properties such as shading and border treatments do not translate directly to DITA XML.

For presentational properties, the standard way would be to have XSLT add shading (for example) based on a row count or other content-independent table property.

For semantic properties, you would have to identify in FrameMaker the cells that require treatment. If you use a dedicated FrameMaker format for the content of those cells, you could assign an `outputclass` value to that format via `[StyleCellAttribute]`; see §24.4.6 [Specifying attributes for table cells](#) on page 738. Alternatively, you could use **CellOutputclass** markers; see §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.

15.6.5.5 Converting table titles to DITA XML

Best practice is to include the table title as part of the table itself in FrameMaker. **Mif2Go** handles such table titles automatically. If you have positioned the title as a separate element before the table in FrameMaker, consider moving the title inside the table, using the FrameMaker *Table Designer* to make the table title accessible. However, if you need table titles to remain separate from tables, you can instruct **Mif2Go** to wrap both title and table in a `<section>` element.

To wrap both a table and its preceding title in a `<section>` element:

```
[DITAParaTags]
TableTitleFmt = title

[DITAParents]
TableTitleFmt = section

[DITAFirst]
TableTitleFmt = section
```

15.6.6 Converting tables used only as image containers

If you use tables to hold images in FrameMaker, and the table title serves as the image caption, you can assign **Mif2Go** table type `figure` to the FrameMaker table format. Table type `figure` is an alias for table type `simple`. Because DITA requires a `<fig>` element to have its `<title>` element (the image caption) *before* the image, for table type `figure` **Mif2Go** does the following:

- uses the content of the table title for the figure `<title>` element
- places the generated `<simplatable>` after the figure `<title>` element
- wraps both in a `<fig>` element.

This keeps the images packaged more or less the way they are in your FrameMaker document. For example:

```

<fig id="ab998633"><title>Project configuration</title>
<simplatable id="ab998614">
<strow>
<stentry><image href="projcfg.gif" placement="inline"/></stentry>
</strow>
</simplatable>
</fig>

```

To map a table format (for example, *FigTable*) to **Mif2Go** table type figure:

```

[DITATables]
FigTable = figure

```

Also map the table title format (for example, *FigTitle*) to the `<title>` element:

```

[DITAParaTags]
FigTitle = title

```

To ensure that the title and table are properly wrapped in a `<fig>` element, specify `<fig>` as the parent of the title format:

```

[DITAParents]
FigTitle = fig

```

As an alternative, if you do not need the table itself in DITA, you can remove the table coding instead:

```

[DITATables]
FigTable = strip

```

See also:

§15.6.7 [Omitting table coding entirely](#) on page 515

§15.7 [Specifying options for images in DITA XML](#) on page 516

15.6.7 Omitting table coding entirely

If your reason for using a particular table format in FrameMaker is strictly presentational, you can have **Mif2Go** omit the table coding for that format, and instead just enclose the table title (if any) and table content in the parent element of the table.

To omit table coding, assign table type `strip` to the table format. **Mif2Go** wraps the content in the parent element assigned to the paragraph format that holds the content. If the table has a title, that title becomes the `<title>` element, and its parent is used as the wrapper; in this case, also map the table title format to the `<title>` element.

For example:

```

[DITATables]
Holder = strip

[DITAParaTags]
HoldTitle = title

[DITAParents]
HoldTitle = section

```

If your document has several tables in a row that are all assigned table type `strip`, you might want the elements they contain to be merged under a single parent element. To prevent the parent element from closing automatically after each table:

```

[Tables]
; CloseStrippedTables = Yes (default) or No (allow elements started
; in stripped tables to remain open until closed for other reasons)
CloseStrippedTables = No

```

When `CloseStrippedTables=No`, you must make sure the parent element gets closed some other way after the last stripped table in each group.

15.7 Specifying options for images in DITA XML

In this section:

- §15.7.1 [Designating ancestors for <image> and <fig> elements](#) on page 516
- §15.7.2 [Specifying what to include in a <fig> wrapper](#) on page 517
- §15.7.3 [Omitting size attributes from images for DITA output](#) on page 518
- §15.7.4 [Providing alternate text for images](#) on page 518
- §15.7.5 [Including MathFullForm equations in <alt> elements](#) on page 518
- §15.7.6 [Including the original image DPI as an attribute](#) on page 518
- §15.7.7 [Understanding why images might look incorrectly scaled](#) on page 519

15.7.1 Designating ancestors for <image> and <fig> elements

To specify the ancestor elements **Mif2Go** must use to wrap <image> and <fig> elements:

```
[DITAOptions]
; ImageParents = parents for <image> tags, whether wrapped in <fig>
; or not; default none (use content model), may include sets from
; [DITAElementSets].
ImageParents = list of parent elements
```

List ancestors in hierarchical order; see §15.5.2 [Designating DITA ancestor elements](#) on page 502. You can include element sets, as well as single elements; see §15.5.5 [Specifying alternate ancestries for the same element](#) on page 504. If you do not specify any ancestor elements, **Mif2Go** picks the first valid element listed in the content model, which might not be what you had in mind.

Note: Do not include `fig` either in the list for `ImageParents` or in an element set in that list.

For example, suppose you want most of your images wrapped in <section>, except for those that occur in paragraphs that are mapped to <example>:

```
[DITAOptions]
ImageParents = $iparents

[DITAElementSets]
$iparents = section example
```

To specify ancestry for a single <image> element or a discrete group of <image> elements, assign the parent name or parent set name to the graphic ID of the image (see §5.3 [Identifying files and objects](#) on page 117), or to the graphic group ID (see §23.5.1.4 [Creating named groups of graphics](#) on page 710). For example, to make sure icons in table cells have <entry> as a parent:

```
[GraphGroup]
ab01f853 = alerts
ab012c13 = alerts
ab00b5d3 = alerts

[DITAIImageParents]
; image ID (may be group) = parents to be used for image/fig element.
alerts = entry
```

You can make a single `[DITAIImageParents]` setting in an `HTMLConfig` marker, also; see §33.2.2 [Overriding settings with configuration markers](#) on page 921.

*Sequence
matters in
element sets*

Although **Mif2Go** knows which elements are valid within other elements, **Mif2Go** has no idea at all about required sequences of elements. For example, if you set:

```
[DITAElementSets]
$iparents = conbody section entry example context choice
```

Mif2Go will always choose example over context when in `<taskbody>`. Where the image is valid in both `<context>` and `<example>`, **Mif2Go** lacks any real criterion for choosing one over the other. Instead, **Mif2Go** selects, from the list of candidates, the first element that is valid as a parent of the `<image>` element.

In this example, if more of your images belong in `<context>`, you could set:

```
[DITAElementSets]
$iparents = conbody section entry context example choice
```

and then use `[DITAImageParents]` for the lesser number of images that should be in `<example>`.

15.7.2 Specifying what to include in a `<fig>` wrapper

When **Mif2Go** wraps image and title in a `<fig>` element, by default **Mif2Go** closes the `<fig>` element before moving on to the following content. To direct **Mif2Go** to include in `<fig>` any following elements that are valid:

```
[DITAOptions]
; CloseFigAfterImage = Yes (default) or No (leave fig open for more)
CloseFigAfterImage = No
```

By default, **Mif2Go** wraps all contiguous images and their titles in a single `<fig>` element. To make sure each of a series of images is wrapped in its own `<fig>` element:

```
[DITAOptions]
; MultiImageFigures = Yes (default)
; or No (allow only one image in a fig)
MultiImageFigures = No
```

When an unstructured FrameMaker document includes several images in a row with only their titles in between, by default **Mif2Go** assumes that these titles follow their respective images. To specify that figure titles precede their images instead:

```
[DITAOptions]
; FigureTitleStartsFigure = No (default, title is below image),
; or Yes (title is above image)
FigureTitleStartsFigure = Yes
```

To prevent an image in an anchored frame from being wrapped in a `<fig>` element, assign the `NoFig` format property to the anchor paragraph format. For example:

```
[HTMLParaStyles]
; NoFig is used in DITA for a graphic anchor para to prevent wrapping
; of the image inside it in a fig tag.
GraphAnchor = NoFig
```

This works only if the FrameMaker anchor format is used consistently for images that should not be wrapped, and not for any that should be wrapped.

To make sure images with one particular FrameMaker anchor format are wrapped, when the rest are not (for example):

```
[HTMLParaStyles]
; Figure is used in DITA for a graphic anchor para to ensure wrapping
; of the image inside it in a fig tag.
SpecialGraphAnchor = Figure
* = NoFig
```

See §4.6 [Using wildcards in configuration settings](#) on page 106.

15.7.3 Omitting size attributes from images for DITA output

To eliminate width and height attributes from images:

```
[Graphics]
; GraphScale = Yes (default) to put out width and height attributes,
; or No to eliminate them all
GraphScale = No
```

If you do not specify any setting for GraphScale, width and height attributes are included.

15.7.4 Providing alternate text for images

To provide alternate text for an image in DITA, you need an `<alt>` tag. Instead of using a **GraphAlt** marker, which causes the content to become an attribute, you have to insert the text via the FrameMaker *Object Attributes* dialog for the anchored frame; see §31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896. **Mif2Go** produces `<alt>` tags if and only if you use the FrameMaker *Object Attributes* dialog. Place the text in the **Alternate:** box under **Text Attributes**. Do not place the text in the **Defined Attributes:** box under **New or Changed Attribute**; if you do, the content will become the attribute, which is deprecated for DITA.

15.7.5 Including MathFullForm equations in `<alt>` elements

When **Mif2Go** encounters in your FrameMaker document an equation, which is a MathFullForm object, **Mif2Go** generates a DITA `<image>` element for the equation. You can also have the original MathFullForm included as the content of the `<alt>` tag.

To include MathFullForm equations in `<alt>` tags:

```
[HTMLOptions]
; MathFullForm = No (default) or Yes: include MathFullForm objects
; in DITA <alt> tags for equations.
MathFullForm = Yes
```

See Figure 5.9.1 [Understanding how equations are processed](#).

15.7.6 Including the original image DPI as an attribute

If you are using DITA-FMx (see §15.2.4 [Specifying DITA version](#) on page 480), you might want to provide a value for `fmdpi: <dpi>` for the images in your document.

Mif2Go can pick up the DPI value of each image from FrameMaker, and include that value in an `@otherprops` attribute for use with DITA-FMx.

To include image DPI values in `@otherprops` for `<image>` elements:

```
[DITAOptions]
; UseOtherpropsDPI = No (default), or Yes (use FrameMaker image
; DPI values for DITA-FMx @otherprops fmdpi values)
UseOtherpropsDPI = Yes
```

When `UseOtherpropsDPI=Yes`, **Mif2Go** includes the original FrameMaker DPI value in `@otherprops` attribute `fmdpi` for each `<image>` element. For example:

```
<image id="z108x11a50b.gif" href="LinkBttn.gif" otherprops="fmdpi:96">
```

The value of the `@otherprops fmdpi` attribute is not affected by the setting for GraphDPI; see §23.9.4 [Specifying image resolution for exported graphics](#) on page 721.

15.7.7 Understanding why images might look incorrectly scaled

If you use FrameMaker version 8 to open a DITA file generated by **Mif2Go**, images might appear to be oversized or stretched out. This is because FrameMaker 8 assumes that image width and height attributes are expressed in points (72/in) rather than in pixels (96/in). FrameMaker 8 implements the DITA 1.0 specification, which does not specify the units of measure to be used for the width and height attributes in the `<image>` tag, nor does it provide a way to specify those units.

According to the DITA 1.1 specification, the default unit of measurement for width and height attributes in the `<image>` tag is pixels. The specification allows any of several suffixes for these attributes, including `px` for pixels and `pt` for points. Unfortunately, FrameMaker treats the attribute values as points, even if they have a `px` suffix.

Presumably this will be fixed when FrameMaker supports the DITA 1.1 specification.

Meanwhile, to force the sizes to points, and label them so that they remain correct for DITA 1.1:

```
[Graphics]
; This is effective for DITA only:
; UsePtSuffix = No (default, unless [DITAOptions]FM8Import=Yes),
; or Yes (FM8Import default, set ConversionDPI to 72 so that FM8
; interprets the size correctly, and include "pt" in the width and
; height attributes to specify that the values are in points)
UsePtSuffix = Yes
```

This setting forces the size to come out in points, because there are 72 points per inch.

Note: `UsePtSuffix=Yes` overrides `UsePxSuffix=Yes`; see §23.9.5 [Specifying px units for graphics sized in pixels](#) on page 722.

15.8 Organizing DITA topics

Unless you allow **Mif2Go** to split each FrameMaker file in your document into separate DITA topic files, topics must be either nested or wrapped in a top-level `<dita>` element.

In this section:

- §15.8.1 [Understanding when to split, nest, or wrap DITA topics](#) on page 519
- §15.8.2 [Splitting FrameMaker files into DITA topic files](#) on page 520
- §15.8.3 [Renaming DITA topic files](#) on page 520
- §15.8.4 [Nesting DITA topics in unsplit files](#) on page 521
- §15.8.5 [Wrapping DITA topics in a top-level `<dita>` element](#) on page 521

See also:

- §16.2 [Configuring DITA ditamaps](#) on page 539

15.8.1 Understanding when to split, nest, or wrap DITA topics

Normally **Mif2Go** splits each FrameMaker file into individual DITA XML output topic files. Although you can choose to generate a single DITA XML file from each FrameMaker chapter file, this is not recommended. Unless all topics in such a monolithic file are of the same topic type and are embedded in a single top-level topic, **Mif2Go** wraps them in a top-level `<dita>` element. The `<dita>` element is an alternative to `<map>`, and is meant to support legacy documents.

If you do not split FrameMaker files into topics, unless you are using the Leximation DITA-FMx plug-in for re-import into FrameMaker, the topics from each FrameMaker file

must be either nested or wrapped to allow topic IDs in map references to those topics. However, if you nest topics, you lose most of the DITA reusability feature. Nesting applies only to multi-topic files.

In general, nesting topics is a Very Bad Idea. It makes re-use much harder. If instead you keep each real topic (no containers!) in a separate file, and assemble the files with ditamaps, you have a much more flexible and capable system. There is no benefit to nested topics, because you can aggregate the topics exactly the same way with a map as with a container; see §16 [Configuring DITA maps](#) on page 539.

Mif2Go forces wrapping of topics in the following circumstances:

- When `NestTopicFiles=No` and `SplitTopicFiles=No`.
- When `NestTopicFiles=Yes`, but map levels are such that the first topic in a file does not nest all the other topics in the file.
- When `WrapTopicFiles=No`, but the file contains more than one topic type.

15.8.2 Splitting FrameMaker files into DITA topic files

By default, **Mif2Go** splits each FrameMaker file into individual DITA XML topic files, one file per topic. To have **Mif2Go** generate instead a single multi-topic DITA XML file from each FrameMaker file (*not recommended*):

```
[DITAOptions]
;SplitTopicFiles = Yes (default, each topic is a file)
; or No (wrap all topics in chapter together with <dita> tag)
SplitTopicFiles = No
```

When `SplitTopicFiles=Yes`, **Mif2Go** splits each chapter file on topic boundaries identified by FrameMaker paragraph formats; see §15.9.1 [Designating starting points for DITA topics](#) on page 522. A topic ends at the start of the next topic, or at the end of the FrameMaker file. When you split a FrameMaker file into individual DITA topic files, the hierarchical relationship among those topics is made explicit in the map levels assigned to those topics; see §16.2.2 [Specifying topic levels in ditamaps](#) on page 544.

When `SplitTopicFiles=No`, unless you nest the topics from each FrameMaker file so there is just one top-level topic, **Mif2Go** wraps all topics within a top-level `<dita>` element. See §15.8.4 [Nesting DITA topics in unsplit files](#) on page 521.

15.8.3 Renaming DITA topic files

Mif2Go assigns each split topic file (except the first) a base name that consists of the FrameMaker chapter FileID followed by the ObjectID of the paragraph that caused the split; see §5.3 [Identifying files and objects](#) on page 117. This naming method guarantees that output file names will be unique; see §18.4.1 [Understanding how split and extract files are named](#) on page 593. You can specify a topic-type prefix for each file name, but do not try to rename files outside of **Mif2Go**, unless you use a tool designed to both fix all interfile references and maintain uniqueness of names.

In this section:

- §15.8.3.1 [Prefixing split-file names to identify DITA topic type](#) on page 520
- §15.8.3.2 [Renaming DITA topic files with FrameScript](#) on page 521

15.8.3.1 Prefixing split-file names to identify DITA topic type

When you split FrameMaker files so that each DITA topic is in its own `.dita` file, you can specify a prefix to each file name to identify the topic type. For example:

```
[DITATopicFileNamePrefix]
; toptype = prefix to file names
concept = c_
task = t_
reference = r_
glossary = g_
```

In this example, a file containing a topic of type `topic` would *not* get a prefix.

15.8.3.2 Renaming DITA topic files with FrameScript

For a way to achieve human-readable DITA topic file names, see Rick Quatro's solution:
<http://frameautomation.com/2010/03/24/renaming-dita-map-topics/>

From Rick's Web site:

The script proposes new topic names, based on the `navtitle` value of the `topicref`. You can have the script automatically use the new names, or you can use a “semi-automatic mode” where the old names and proposed new names are written to an Excel file. Then you can use Excel to fine-tune the new names. A second script quickly applies the spreadsheet names to the DITA map and corresponding topic files.

Note: This method would not work for references between topics in different maps.

15.8.4 Nesting DITA topics in unsplit files

To nest the topics in a DITA XML output file:

```
[DITAOptions]
; NestTopicFiles = No (default, recommended),
; or Yes (if SplitTopicFiles=No, nest the topics in the <dita> file
; per their [DITAMapLevels]).
NestTopicFiles = Yes
```

When `NestTopicFiles=Yes`, topics are nested in each file according to their map levels. To make the nesting valid using DITA Composite, in the following cases **Mif2Go** wraps all topics within a top-level `<dita>` element:

- map levels are such that the first topic in a file does not nest all the other topics; see §16.2.2 [Specifying topic levels in ditamaps](#) on page 544
- the file contains topics of more than one basic topic type (`topic`, `concept`, `task`, or `reference`).

When `NestTopicFiles=No`, if `SplitTopicFiles=No`, **Mif2Go** wraps all topics from a FrameMaker chapter file within a top-level `<dita>` element according to their map levels, in a single DITA XML output file, regardless of the setting for `WrapTopicFiles`.

See §15.8.5 [Wrapping DITA topics in a top-level <dita> element](#) on page 521.

15.8.5 Wrapping DITA topics in a top-level <dita> element

If you split FrameMaker files into separate topic files, and you use topic IDs in map references to those topics, for re-import into FrameMaker 8 each topic file must be wrapped in a `<dita>` element (unless you are using the Leximation DITA-FMx plug-in). Otherwise, FrameMaker version 8 will not be able to open the files.

To wrap the topics in each file in a `<dita>` element:

```
[DITAOptions]
; WrapTopicFiles = No (default), or Yes (needed to allow TopicIDs for
; Frame8 import unless DITA-FMx is installed).
WrapTopicFiles = Yes
```

When `WrapTopicFiles=Yes`, all topics are wrapped in a `<dita>` element in each DITA XML output file. Within this wrapper, any topic type except `glossary` can nest topics of any other type.

When `WrapTopicFiles=No`, topics are wrapped in `<dita>` tags only if required by a mixed-type nesting condition, or if `SplitTopicFiles=No` and the first topic in a file does not nest all the other topics. When topics are not wrapped, each topic type (except `glossary`) can nest only instances of the same type; a `glossary` topic cannot nest any topic. If any topic type nests other topic types, **Mif2Go** forces `WrapTopicFiles=Yes`. If a `glossary` topic tries to nest any topic, **Mif2Go** promotes the nested topic to a sibling. See §15.8.4 [Nesting DITA topics in unsplit files](#) on page 521.

You might need to set `WrapTopicFiles=Yes` if you are mixing single-topic DITA files with DITA content from files that contain multiple topics in a `<dita>` wrapper, perhaps from another source (so you cannot make them single-topic files). You must use topic IDs for map references to the multi-topic files, so you would have to wrap the single-topic files also, to make the topic IDs acceptable there.

When `[DITAOptions]FM8Import=Yes`, **Mif2Go** changes the default value of `WrapTopicFiles` to `Yes`; see §15.2.6 [Ensuring FrameMaker 8 import compatibility](#) on page 481.

Note: When you use topic IDs in map `href` attributes, chapter files produced by FrameMaker version 8 from those topic files have names of the form `TopicFileName#TopicID.fm`. See §16.2.1.7 [Excluding topic IDs for FrameMaker 8 import](#) on page 543.

15.9 Configuring DITA topics

Mif2Go delimits DITA topics in your FrameMaker document according to paragraph formats you identify as `<title>` elements at DITA level 1.

In this section:

- §15.9.1 [Designating starting points for DITA topics](#) on page 522
- §15.9.2 [Specifying the DITA topic type](#) on page 524
- §15.9.3 [Specifying the ID for a DITA topic](#) on page 526
- §15.9.4 [Adjusting DITA topic IDs generated from file names](#) on page 526
- §15.9.5 [Specifying alternate titles for a DITA topic](#) on page 526
- §15.9.6 [Omitting a DITA topic from the TOC](#) on page 527

15.9.1 Designating starting points for DITA topics

Mif2Go bases starting points for DITA topics on the occurrence in your FrameMaker document of paragraph formats that have certain configuration settings. By default, **Mif2Go** also treats the very first paragraph format in each FrameMaker file as the start of a DITA topic.

In this section:

- §15.9.1.1 [Identifying starting elements for non-glossary topics](#) on page 523
- §15.9.1.2 [Identifying the starting element for glossary topics](#) on page 523
- §15.9.1.3 [Preventing the first paragraph format from starting a topic](#) on page 523

15.9.1.1 Identifying starting elements for non-glossary topics

For topics of all built-in DITA types except `glossary`, the required starting element is `<title>`. **Mif2Go** identifies a topic start by the paragraph format mapped to `title` in `[DITAParaTags]`.

To designate a paragraph format as a DITA topic start:

1. Unless the format is already named *Title*, map the format to the `<title>` element:

```
[DITAParaTags]
ParaFmt = title
```

See §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487.

2. Assign the format DITA level 1:

```
[DITALevels]
ParaFmt = 1
```

See §15.5.13 [Specifying DITA element levels](#) on page 509.

15.9.1.2 Identifying the starting element for glossary topics

For `glossary` topics (DITA version 1.1+ only), the required starting element is `<glossterm>`. Unless the default topic type is `glossary`, you must tell **Mif2Go** that the topic start is the paragraph format mapped to `glossterm` in `[DITAParaTags]`.

You do not have to specify parents for glossary elements, because `<glossterm>` and `<glossdef>` can have only `<glossentry>` as parent; and you do not have to specify an element level for the format mapped to `glossterm`, because it will always be level 1.

*Glossary in a
separate file*

If the glossary for your document is in a separate FrameMaker file, not mixed with other types of topics, create a file-specific configuration file `glossfile.ini`, and include in it the following setting:

```
[DITAOptions]
DefTopic = glossary
```

See §15.9.2.2 [Specifying a default DITA topic type](#) on page 525.

Setting the default topic type to `glossary` tells **Mif2Go** that the topics in the current file start with the paragraph format mapped to `glossterm` in `[DITAParaTags]`:

```
[DITAParaTags]
ParaFmt = glossterm
```

See §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487.

*Glossary mixed
with other topics*

If the glossary for your document is in a FrameMaker file that includes other topic types, you must make the starting paragraph format for the glossary topic different from the starting formats for all other topic types in the file; and you must assign topic type `glossary` to that format in `[DITATopics]`. For example:

```
[DITATopics]
; Every GlossaryTerm paragraph begins a glossary topic:
GlossaryTerm = glossary

[DITAParaTags]
; Every glossary topic begins with a <glossterm> element:
GlossaryTerm = glossterm
Definition = glossdef
```

15.9.1.3 Preventing the first paragraph format from starting a topic

To prevent **Mif2Go** from forcing the first paragraph format in a FrameMaker file to become a topic start:

```
[DITAOptions]
; ForceStartTopic = Yes (default, make the format of the first para a
;   topic start with tag "title" at level 1 and parent "topic"), or No.
ForceStartTopic = No
```

If the first paragraph format in a file is *not* assigned DITA level 1, or is *not* implicitly or explicitly mapped to the <title> element:

- When ForceStartTopic=Yes, **Mif2Go** forces these settings, overriding any other mapping or level assignment; as a consequence, all subsequent paragraphs with the same format in the same file also start topics.
- When ForceStartTopic=No, **Mif2Go** allows the paragraph to produce invalid DITA.

Best practice is to use the default setting (ForceStartTopic=Yes), and make sure the first paragraph format in each FrameMaker file is mapped to title and assigned level 1.

However, if your FrameMaker document uses a table to hold the chapter title, you might have to set ForceStartTopic=No, and make sure the table anchor paragraph does not get converted. For example, if you use paragraph format *TableAnchor* for the anchor, set:

```
[HTMLParaStyles]
TableAnchor = Delete
```

If there is a second such table in the chapter file, also set:

```
[Tables]
AllowTbSplit = Yes
```

To omit the table tags for chapter-title tables:

```
[DITATables]
ChapTitleTableFmt = Strip
```

See §15.6.7 [Omitting table coding entirely](#) on page 515.

15.9.2 Specifying the DITA topic type

Mif2Go provides three ways to indicate the type of a topic: specify a default type, assign a type to a paragraph format, or insert a marker in the topic with the name of the type.

In this section:

§15.9.2.1 [Understanding DITA topic type assignment precedence](#) on page 524

§15.9.2.2 [Specifying a default DITA topic type](#) on page 525

§15.9.2.3 [Specifying the DITA topic type with a paragraph format](#) on page 525

§15.9.2.4 [Specifying the DITA topic type with a marker](#) on page 525

15.9.2.1 Understanding DITA topic type assignment precedence

The default type for every topic **Mif2Go** generates from your FrameMaker document is type concept. You can specify a different default, assign a topic type to a paragraph format, or use a marker to specify the type of an individual topic. [Table 15-2](#) shows the precedence of topic type assignment methods.

Table 15-2 Precedence of DITA topic type assignment methods

Precedence	Method	Reference
Highest	DITAtopic marker	15.9.2.4
Intermediate	FrameMaker paragraph format	15.9.2.3
Lowest	Default topic type	15.9.2.2

If you specify a custom specialized topic type (a type other than `topic`, `concept`, `task`, `reference`, `glossary`, or `map`), you must provide a separate content-model configuration file for the specialized type, named `DITAtopictype.ini`; see §32 [Working with content models](#) on page 905.

15.9.2.2 Specifying a default DITA topic type

By default, every topic generated is of type `concept`. However, you can specify a different topic type as the default: `topic`, `task`, `reference`, `glossary`, or a custom type.

To specify a different default topic type:

```
[DITAOptions]
; DefTopic = name of topic type to use, default "concept"
DefTopic = topic
```

You can override the default topic type with a **DITATopic** marker in the topic, or by assigning a different topic type to a FrameMaker paragraph format used in the topic.

To specify a different default topic type for a given FrameMaker chapter, include the `DefTopic` setting in a chapter configuration file named for the FrameMaker file; see §33.1.1 [Providing configuration files for individual chapters](#) on page 919.

15.9.2.3 Specifying the DITA topic type with a paragraph format

To assign a DITA topic type to a FrameMaker paragraph format (for example):

```
[DITATopics]
; Frame para format = suggested topic type to use, if no DITATopic
; marker found.
Step = task
Syntax = reference
GlossItem = glossary
```

Assign a topic type to any paragraph format for which at least one of the following is true:

- The format is specific to a topic type other than the default topic type; see §15.9.2.2 [Specifying a default DITA topic type](#) on page 525.
- The format marks a transition from one topic type to another, even if the new topic is the default topic type.
- The format starts a topic for which the starting element is not `<title>`; for example, topics of type `glossary` (see §15.9.1.2 [Identifying the starting element for glossary topics](#) on page 523). If necessary, modify your FrameMaker document to use a dedicated format for such topic starts.

You can override the assigned topic type with a **DITATopic** marker placed in the topic.

If **Mif2Go** encounters in FrameMaker multiple paragraph formats in the same topic with different topic type assignments in `[DITATopics]`, only the topic type assigned to the last paragraph format encountered before the end of the topic is considered.

If **Mif2Go** encounters in FrameMaker a paragraph format that has a topic type assignment in `[DITATopics]` and a conflicting element mapping in `[DITAParaTags]`, the topic type takes precedence, and the tag instance is flagged as an error.

15.9.2.4 Specifying the DITA topic type with a marker

You can use a **DITATopic** marker to override the default topic type for a given topic, and also any topic type assigned via paragraph format. The content of a **DITATopic** marker is the name of the topic type. Insert the **DITATopic** marker anywhere in the content of the topic.

15.9.3 Specifying the ID for a DITA topic

To give a topic an ID, do one of the following:

- Place a **DITATopicID** marker in the topic; the content of the **DITATopicID** marker is the topic ID.
- Place a **FileName** marker in the topic; the marker content specifies both the topic ID and the base file name of the split file that contains the topic (see §34.8.3 [Using custom markers to name output files](#) on page 947).

In the absence of either marker, the default topic ID is the base split-file name, adjusted as for CSS class names; see §15.9.4 [Adjusting DITA topic IDs generated from file names](#) on page 526.

If you do not split files (see §15.8.2 [Splitting FrameMaker files into DITA topic files](#) on page 520), nor insert markers for topic IDs, **Mif2Go** makes up an ID for each embedded topic after the first (which uses the base file name). These generated IDs are of the form `topic2`, `topic3`, and so forth. This is not recommended practice.

To specify a default ID other than the base split-file name (but only for the *first* topic in a FrameMaker file), include the following option in a chapter-specific configuration file (see §33.1.1 [Providing configuration files for individual chapters](#) on page 919) located in the project directory:

```
[DITAOptions]
; TopicID = id for topic, default is base file name
TopicID = someid
```

You can override the default ID with a **DITATopicID** marker, or with a **FileName** marker.

15.9.4 Adjusting DITA topic IDs generated from file names

By default, **Mif2Go** uses the base name of the output file that contains a DITA topic as the ID for that topic. Because topic IDs may not contain spaces, by default **Mif2Go** removes any spaces in the ID, and makes the ID all lowercase. You can specify other treatments: a character to replace each space, remove underscores, keep original case.

To adjust topic IDs generated from file names:

```
[DITAOptions]
; These are used only when generating a TopicID from the file name:
; DITATopicIDSpaceChar = char to replace spaces, default none
; (remove space)
DITATopicIDSpaceChar=_
; DITATopicIDUnderscore = Yes (default, keep underscores)
; or No (remove)
DITATopicIDUnderscore=Yes
; DITATopicIDLowerCase = Yes (default, make all lower)
; or No (retain original)
DITATopicIDLowerCase=Yes
```

If you have FrameScript installed on your system, you can use a script to create more “human readable” topic names for DITA output. See:

<http://frameautomation.com/2010/03/24/renaming-dita-map-topics/>

15.9.5 Specifying alternate titles for a DITA topic

To include an alternate title for a topic, you can use either of the following:

[Dedicated paragraph format](#)

[FrameMaker marker](#)

The text of an alternate title, whether provided via paragraph format or marker, appears as follows:

Navigation title: `navtitle` attribute in the map `<topicref>`

Search title: `<searchtitle>` element in the `<topicmeta>` of the map's `<topicref>`

In addition, both appear as elements in the topic itself, in a `<titlealts>` block immediately following the `<title>` element. **Mif2Go** provides the `<titlealts>` wrapper for alternate titles.

*Dedicated
paragraph format*

To use a dedicated paragraph format for an alternate title, place a paragraph in that format after the main title paragraph, and map the format to the alternate-title element. For example:

```
[DITAParaTags]
; Frame para format (wildcards OK) = DITA element
NavHead = navtitle
Search = searchtitle
```

See §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487.

*FrameMaker
marker*

To use a marker for an alternate title, insert a **DITANavTitle** marker, a **DITASearchTitle** marker, or both, in the `<title>` paragraph. Make the content of the marker the text of the alternate title, which becomes the content of the `navtitle` attribute or `<searchtitle>` element.

15.9.6 Omitting a DITA topic from the TOC

To omit any reference in the TOC to a topic whose title would otherwise appear there, insert a marker of type **DITANoTOC** in `<title>` paragraph of the topic. The **DITANoTOC** marker will have the following effects on the map `<topicref>` to the topic:

- The `<topicref>` will include `toc="no"`
- The `navtitle` for the `<topicref>` will be suppressed.

No content is required in the **DITANoTOC** marker.

15.10 Configuring cross references and links for DITA

In this section:

§15.10.1 [Understanding how Mif2Go converts cross references](#) on page 527

§15.10.2 [Specifying an outputclass for cross-reference wrappers](#) on page 528

§15.10.3 [Linking to elements below topic level](#) on page 528

§15.10.4 [Retaining cross-reference content in <xref> elements](#) on page 528

§15.10.5 [Omitting <xref> elements from footnotes](#) on page 529

§15.10.6 [Overriding <xref> attribute values](#) on page 529

15.10.1 Understanding how Mif2Go converts cross references

Mif2Go converts FrameMaker cross references and hypertext links to DITA `<xref>` elements. DITA allows cross references to `<topic>` (including each basic type), `<section>` (including `<example>` and `<refsyn>`), `<table>`, `<fig>`, `<fn>`, and ``. No other elements. **Mif2Go** provides an ID for each instance of each of these elements, if a suitable ID is not already present.

When a `<xref>` tag appears in a context where it is not valid, such as in a title, **Mif2Go** automatically wraps the `<xref>` in a `<ph>` element, and assigns an `outputclass` attribute to the wrapper; see §15.10.2 [Specifying an outputclass for cross-reference wrappers](#) on page 528.

To provide a link destination for target elements that do not already contain a **DITAElemID** marker (see §15.4.6.1 [Specifying a value for the id attribute](#) on page 495), **Mif2Go** makes the ID of the target element the content of the first **newlink** marker in the element; or, in the absence of **newlink** markers, the numeric ID of the cross-reference marker.

15.10.2 Specifying an outputclass for cross-reference wrappers

When **Mif2Go** encounters a cross reference, index marker, or footnote reference in a context where an `<xref>` tag would be invalid, the `<xref>` gets wrapped in a `<ph>` element. These **Mif2Go**-generated `<ph>` wrapper elements need an `outputclass` attribute. The default `outputclass` attribute names for cross-reference, index-term, and footnote wrappers are as follows:

```
[DITAOptions]
; XrefWrapClass = outputclass to use for generated ph elements that
; wrap xrefs where they would otherwise be invalid
XrefWrapClass = phxref
; IndexWrapClass = outputclass to use for generated ph elements that
; wrap indexterms where they would otherwise be invalid
IndexWrapClass = phindex
; FootnoteWrapClass = outputclass to use for generated ph elements
; that wrap footnotes where they would otherwise be invalid
FootnoteWrapClass = phfoot
```

You can specify other `outputclass` attribute names.

15.10.3 Linking to elements below topic level

Normally, **Mif2Go** can use a FrameMaker **newlink** marker or cross-reference marker to create a link for an element ID. However, to provide a destination for a link to a target element that is below topic level, in some cases you might have to insert a **DITALinkElemID** marker in text just before the link, and a **DITAElemID** marker (or **DITAParentID** marker) in the target text.

The content of the **DITALinkElemID** marker must match the content of the **DITAElemID** marker or **DITAParentID** marker (see §15.4.6.1 [Specifying a value for the id attribute](#) on page 495). This is to ensure the correct link target in cases where the built-in rules to determine the target element ID provide the wrong choice or none at all. **Mif2Go** places the **DITALinkElemID** marker content after the target topic ID in the `href` attribute of the `<xref>` element.

15.10.4 Retaining cross-reference content in `<xref>` elements

By default, **Mif2Go** does not include the text of FrameMaker cross references in DITA `<xref>` elements. However, you might want to retain the text, especially if you anticipate round-tripping between DITA and FrameMaker, or using DITA output as a stepping stone from unstructured to structured FrameMaker.

To retain the content of FrameMaker cross references in DITA `<xref>` elements:

```
[DITAOptions]
; KeepXrefText = No (default) or Yes (retain content of xref in DITA
; as is done for hyperlinks; prevents updating of xref during later
```

```
; processing by the DITA Open Toolkit).
KeepXrefText = Yes
```

A disadvantage of retaining cross-reference content is that doing so prevents you from updating the text of the reference later. When the `<xref>` element is empty, it is populated from the content of the referenced item.

15.10.5 Omitting `<xref>` elements from footnotes

The way DITA handles footnotes results in footnotes with IDs not getting callouts unless an `<xref>` element is added to provide the callout. However, the DITA Open Toolkit is inconsistent on this point. The HTML transform wants the `<xref>` element, but the PDF2 transform does not.

To exclude `<xref>` elements from footnotes:

```
[DITAOptions]
; FootnoteXref = Yes (default, comply with spec) or No (indulge bug in
; DITA-OT for pdf2 output using Idiom/RenderX by omitting footnote
; xref)
FootnoteXref = No
```

15.10.6 Overriding `<xref>` attribute values

You can insert markers in your FrameMaker document to change values of the `scope`, `format`, and `type` attributes of individual `<xref>` elements generated from cross-reference and hypertext links.

In this section:

§15.10.6.1 [Specifying the `<xref>` scope attribute](#) on page 529

§15.10.6.2 [Specifying the `<xref>` format attribute](#) on page 529

§15.10.6.3 [Specifying the `<xref>` type attribute](#) on page 530

15.10.6.1 Specifying the `<xref>` scope attribute

By default, for most links **Mif2Go** omits the `scope` attribute of the generated DITA `<xref>` element, so that the value of the `scope` attribute defaults to `local`. However, for links that use a **message URL** hypertextHyperLink marker, **Mif2Go** sets the `<xref>` `scope` to `external`.

Some applications, such as XMetaL, do not support the `#IMPLIED` default of `<xref scope="local">`. If you plan further DITA processing using such an application, you can instruct **Mif2Go** to always make this attribute value explicit:

```
[DITAOptions]
; UseLocalScope = No (default, omit scope attr from xref if not
; specified and href is not to a URL) or Yes (set scope="local" in
; those cases to satisfy applications (XMetaL) that do not respect
; #IMPLIED in the DTD).
UseLocalScope = Yes
```

To specify the `scope` attribute of an individual cross reference or hypertext link in FrameMaker, insert a **DITALinkScope** marker in text just before the link. The content of the marker is the value of the `scope` attribute (usually `peer`) of the generated `<xref>` element.

15.10.6.2 Specifying the `<xref>` format attribute

Mif2Go bases the value of the `<xref>` `format` attribute on the apparent destination of a FrameMaker link. To override this value, insert a **DITALinkFormat** marker in text just

before the link; the content of the marker is the value of the `format` attribute of the generated `<xref>` element. **DITALinkFormat** markers are needed only where **Mif2Go** built-in rules do not produce the correct value.

15.10.6.3 Specifying the `<xref>` type attribute

Mif2Go sets the `type` attribute of most generated `<xref>` elements to the DITA type (root element) of the destination topic. However, for FrameMaker links that use a **message URL** hypertextHyperLink marker, **Mif2Go** omits the `type` attribute.

To specify the `type` attribute of the next cross reference or hypertext link in FrameMaker, insert a **DITALinkType** marker in text just before the link. The content of the marker is the value of the `type` attribute of the generated `<xref>` element. Valid `type` attributes include `li`, `fn`, `fig`, `table`, and `section`.

See also:

§15.4.6.1 [Specifying a value for the `id` attribute](#) on page 495

15.11 Exporting FrameMaker variables to DITA XML

DITA does not really support the concept of variables, other than `conref` attributes. Apparently the truly DITA-proper thing to do is to use `conref` attributes at the `<ph>` level.

In this section:

§15.11.1 [Understanding how Mif2Go represents variables in DITA](#) on page 530

§15.11.2 [Specifying how to treat FrameMaker variables](#) on page 530

§15.11.3 [Treating FrameMaker variables as conrefs](#) on page 531

§15.11.4 [Retaining format properties of user variables in DITA](#) on page 532

15.11.1 Understanding how Mif2Go represents variables in DITA

Mif2Go can incorporate into DITA XML output (and also treat as macro variables) any FrameMaker user variables in your document, as well as a few FrameMaker system variables, provided you reference the names of these variables as described in §28.3.5 [Treating FrameMaker user variables as macro variables](#) on page 801.

Mif2Go supports representing a FrameMaker variable as a `<ph>` element with a `conref` to another `<ph>` element located in a file of such elements, where name and value elements are wrapped in a definition list. **Mif2Go** also supports including user variables as entities, to allow round-tripping using the Leximation DITA-FMx plug-in: the successor to the Frame 7.2 DITA AppPack withdrawn by Adobe.

However, the default treatment is to convert each variable to text; see §15.11.2 [Specifying how to treat FrameMaker variables](#) on page 530.

15.11.2 Specifying how to treat FrameMaker variables

To specify a treatment for FrameMaker user variables and a few FrameMaker system variables in DITA XML:

```
[DITAOptions]
; VariableType = Text (default),
; Entity (for compatibility with DITA-FMx import),
; or Conref (referring to a variable library file).
VariableType = Text
```

This setting affects all FrameMaker user variables in your document and also the following FrameMaker building-block system variables:

- Volume Number*
- Chapter Number*
- Section Number* (FrameMaker version 9+)
- Subsection Number* (FrameMaker version 9+)

When `VariableType=Text` (the default), FrameMaker variables are converted to text before they are included in DITA XML output.

When `VariableType=Entity`, each FrameMaker variable used in a topic becomes an entity. The entity name is the name of the variable, with spaces and any punctuation removed; case is preserved. The entity definition is plain text only; character formatting present in the FrameMaker definition of the variable is omitted. Each topic contains entity declarations only for the entities actually used in that topic. This method allows DITA-FMx to recognize FrameMaker variables on re-import of the generated DITA files.

See also:

§15.11.3 [Treating FrameMaker variables as conrefs](#) on page 531

15.11.3 Treating FrameMaker variables as conrefs

You can make the content of FrameMaker user variables (and some FrameMaker system variables) available as DITA conrefs; see §15.11.2 [Specifying how to treat FrameMaker variables](#) on page 530.

To specify that FrameMaker variables should be treated as DITA conrefs:

```
[DITAOptions]
VariableType = Conref
```

When `VariableType=Conref`, the following settings are effective:

```
[DITAOptions]
; VariableFile = file for referencing variables
VariableFile = ditavars.dita
; VariableTopicID = topic ID for the single topic in the VariableFile
VariableTopicID = varset
; VariableElement = wrapper element for variables,
; valid in a dd element.
VariableElement = ph
; WriteVariableFile = No (default) or Yes (write a file containing all
; variables defined in the Frame file after processing that file).
WriteVariableFile = Yes
```

The `VariableFile` library file for referencing variables contains a single topic, with a single definition list in the `<body>`:

```
<dlentry>
  <dt>varname</dt>
  <dd><ph id="varname">variable content</ph></dd>
</dlentry>
```

Each definition term is the name of a variable with spaces and punctuation removed. The corresponding description has an ID identical to the name, and content identical to the FrameMaker variable definition, except in plain text only.

Re-importing to FrameMaker, you would see (for example):

```
<ph conref="varfile.dita#varsetid/varname"></ph>
```

If the file name and topic ID match your setting, you can be sure that *varname* is a FrameMaker variable, and you can retrieve its definition.

When `WriteVariableFile=Yes`, **Mif2Go** generates a `VariableFile` library file for each FrameMaker file.

*Retaining
variables for an
entire book*

If you are producing DITA output from a FrameMaker book, the `VariableFile` library file will be overwritten for each FrameMaker chapter file until the last. Your variables-turned-conrefs will appear correctly in DITA output after conversion, but only the variables in the last FrameMaker file processed will remain in the `VariableFile` library file. This is because a FrameMaker variable does not have a single book-wide definition; the same variable can be defined differently in every FrameMaker file, or might not be defined at all in some files.

To preserve all the variables in a book, you can do one of the following, depending on whether variables are always defined the same wherever they occur in your FrameMaker book:

Defined the same wherever

Defined differently in some chapters.

*Defined the same
wherever*

To produce one variables library file for the entire book, create a dummy FrameMaker file that defines all the variables, and put it last in the book. Then the last variables file written will be complete, and all the other files will reference it.

*Defined
differently in
some chapters*

To produce a variables library file for a single FrameMaker chapter file, specify a different value of `[DITAOptions]VariableFile` in a separate configuration file named for each FrameMaker file. See §30.4 [Including chapter-specific configuration files](#) on page 855. If only a few chapters have variant definitions, you can provide chapter-specific configuration files only for those chapters, and include at the end of the book a dummy FrameMaker file with all the standard definitions.

Note: You might have an issue with the DITA version of that last dummy file appearing as a chapter in the bookmap.

15.11.4 Retaining format properties of user variables in DITA

If you convert FrameMaker user variables to entities or conrefs, DITA XML cannot retain format properties of those variables, because character formats are converted to tags only in a specific context. XML has no tag that turns off other tags, unless you specify them explicitly; so, for example, `<Default ¶ Font>` has no reasonable meaning in an entity or a conref. If you need the format properties in DITA XML, you must let variables become part of each instance of the text, which is the default, rather than turn them into entities or conrefs; see §15.11.2 [Specifying how to treat FrameMaker variables](#) on page 530.

To retain character format properties for a specific FrameMaker user variable, you must use a **Mif2Go** macro variable of the same name to shadow that user variable; see §5.4.2 [Replacing values of FrameMaker user variables](#) on page 123. Tags must be balanced, as in:

```
[MacroVariables]
TechnologyName = <b>New Technology</b>
```

Even with balanced tags, you will create invalid XML if you use the variable in an area that does not allow `...`, such as nested within another set of `...` tags.

15.12 Converting conditions to DITA attributes

Mif2Go can convert FrameMaker text conditions to attributes in DITA output elements.

In this section:

§15.12.1 [Understanding how Mif2Go converts conditional text](#) on page 533

§15.12.2 [Mapping FrameMaker conditions to element attributes](#) on page 533

§15.12.3 [Disallowing condition conversion for selected elements](#) on page 534

See also:

§13.10 [Converting conditions to HTML attributes](#) on page 446

15.12.1 Understanding how Mif2Go converts conditional text

If a full DITA element (either paragraph or character, block or inline) is conditional, **Mif2Go** sets attributes you designate for the element. If the condition does not apply to the entire element, **Mif2Go** encloses the conditional part in a pair of tags, with the same attributes. By default, **Mif2Go** uses `<ph>`. However, you can specify another tag to use for this purpose:

```
[HTMLOptions]
; ConditionCharTag = tag to interpolate for conditions that affect
; only part of the enclosing element, default ph for DITA.
ConditionCharTag = tagname
```

For example:

```
<ph platform="linux">...</ph>
```

When conditions overlap each other, or overlap inline elements, **Mif2Go** creates a new tag pair for each change, to respect XML no-overlap rules. For example:

```
<p>This paragraph contains <ph product="A">text for </ph>
<i><ph product="A">ProductA </ph><ph product="A B">as well as</ph>
<ph product="B"> text</ph></i><ph product="B"> for ProductB</ph>,
with overlapping conditions and a character format overlapping both,
resulting in five <ph> elements.</p>
```

In addition to text, **Mif2Go** applies conditions to `<table>`, `<image>`, `<xref>`, and `<indexentry>` elements, based on the conditions in effect in FrameMaker at the point of the table or figure anchor, cross reference, or marker.

Mif2Go supports conditional table rows; row condition attributes are not applied to the paragraphs within cells. Likewise, the attributes of block tags are not applied to inline tags enclosed within the block.

Where multiple blocks make up a larger element, such as when a DITA `<title>` element plus some `<p>` elements make up a `<section>`, **Mif2Go** does not push the attributes up to the `<section>` element; they remain on the enclosed block elements.

15.12.2 Mapping FrameMaker conditions to element attributes

To convert conditional text to DITA attributes, you must set FrameMaker **Show/Hide** to show all the conditions for which you want content present; for migration, that would be **Show All**. The conditions shown are identified in DITA output. Content that remains hidden in FrameMaker is not included. Multiple conditions result in multiple conditional attribute values.

To map a FrameMaker condition to an element attribute:


```
[ConditionAttributes]
; Condition name = attributes for elements
CondName= attrname="attrvalue"
```

For DITA, the attribute name is usually one of props, platform, product, audience, or otherprops. For example:

```
[ConditionAttributes]
ProductA = product="A"
ProductB = product="B"
```

15.12.3 Disallowing condition conversion for selected elements

To disallow conversion of conditions to attributes for selected elements:

```
[DITAOptions]
; NoCondAttrs = space-delimited list of elements for which conditional
; attributes (such as props and audience) are disallowed.
NoCondAttrs = element1 element2 ...
```

Usually these are elements derived from <title>. Because <title> and <glossterm> always disallow such attributes, they need not be included in the list. **Mif2Go** includes inside a <ph> nested in the element any attributes that are actually needed. This applies to all attributes mentioned in [ConditionAttributes]; see §15.12.2 [Mapping FrameMaker conditions to element attributes](#) on page 533.

15.13 Marking FrameMaker text insets in DITA

If you reuse FrameMaker text insets extensively, and you do not want to lose this capability when you migrate a document to DITA, you might want text insets to be marked so you can separate them out for further processing. **Mif2Go** can delimit a FrameMaker text inset in DITA XML output by bracketing the content with <data> elements.

To use <data> elements to mark the start and end of each FrameMaker text inset:

```
[DITAOptions]
; TextInsetMark = No (default) or Yes (mark text inset start and end,
; using <data> elements to specify FrameMaker source for the inset)
TextInsetMark = Yes
```

When TextInsetMark=Yes, **Mif2Go** provides two <data> tags, one at each end of the text-inset content, with attributes as follows.

- Starting <data> element:

```
<data
  datatype="text_inset"
  name="insetN"
  value="start"
  href="insets/filename#Bflowname"
  format="fm"
  scope="external"
/>
```

- Ending <data> element:

```
<data
  datatype="text_inset"
  name="insetN"
  value="end"
/>
```

For the name attribute, each inset **Mif2Go** encounters gets an incremental value. The name attributes are numbered beginning with inset1 at the start of each FrameMaker container file.

For the href attribute, if the inset came from a FrameMaker file but did not come from the main flow in that file, **Mif2Go** adds either #B for a body flow or #R for a reference-page flow, followed by the flow tag.

If an inset crosses a topic boundary, the start and end <data> elements will not be in the same topic. **Mif2Go** does not mark the end of an inset at the end of the topic unless the inset really ends there, by design. It would not make sense to re-start the inset in the next topic, because the inset content in that next topic would not begin at the start of the FrameMaker inset file.

Two consecutive FrameMaker text insets might look like this in DITA XML:

```
<p><data datatype="text_inset" name="inset2" value="start"
href="insets/FlowInset.fm#BBeta" format="fm" scope="external" />
This is the "Beta" inset body-page flow.</p>
<p><data datatype="text_inset" name="inset2" value="end" />

<data datatype="text_inset" name="inset3" value="start"
href="insets/FlowInset.fm#RRefInset" format="fm" scope="external" />
This is a reference-page flow, "RefInset".</p>
<p><data datatype="text_inset" name="inset3" value="end" />

This is the container paragraph for the two insets.</p>
```

By default, **Mif2Go** brackets only top-level text insets with <data> elements; if an inset contains another inset, the nested inset is not bracketed. However, you can instruct **Mif2Go** to bracket nested text insets.

To use <data> elements to mark the start and end of each nested text inset:

```
[DITAOptions]
; TextInsetNest = No (default, ignore nested insets)
; or Yes (mark nested insets)
TextInsetNest = Yes
```

When TextInsetNest=Yes, nested text insets would look like this in DITA XML:

```
<p>Next we insert an inset which itself contains an inset:</p>

<p><data datatype="text_inset" name="inset5" value="start"
href="insets/FlowInset.fm" format="fm" scope="external" />
This is the "Alpha" inset body-page flow. In it, we nest this:
<data datatype="text_inset" name="inset6" value="start"
href="insets/SectInset.fm#RFM8Issue" format="fm" scope="external" />
This is the content of the "FM8Issue" nested inset.</p>

<p><data datatype="text_inset" name="inset6" value="end" /></p>

<p><data datatype="text_inset" name="inset5" value="end" />This para
is back in the container. Note the empty para, a result of nesting
an inset at the end of another inset.</p>
```

15.14 Including CSH targets in DITA XML

If your source document includes context-sensitive help targets that you want to include in DITA output for use in further transformations, make sure those targets are in the form of **TopicAlias** markers. **Mif2Go** includes the content of **TopicAlias** markers in DITA output by default. To exclude that content from DITA output:

```
[DITAOptions]
; UseTopicAlias = Yes (default, include in DITA output) or No
UseTopicAlias = No
```

When UseTopicAlias=Yes, **Mif2Go** processes the content of each **TopicAlias** marker into the following:

```
<data name="topicalias" value="IDH_about" />
```

Each such `<data />` element is on a line of its own in the output, placed at the beginning of the next paragraph text (typically the `<title>`).

This format is similar to the DITA-FMx format, but omits the FrameMaker-specific `@datatype`. For example, in DITA-FMx the same CSH target looks like this:

```
<data datatype="fm:marker" name="TopicAlias" value="IDH_about" />
```

See §7.10.2 [Specifying CSH mappings](#) on page 241.

15.15 Overriding DITA settings with markers

You might need to insert markers to override configuration settings for particular DITA topics and elements. **Mif2Go** provides numerous predefined marker types for this purpose, listed in [Table 15-3](#). Most of these marker types are intended to provide ways to cope with unusual situations. If you have a consistent template, and use normal FrameMaker cross references and hypertext links, for the most part you can get by with just configuration settings.

Table 15-3 Predefined marker types for DITA XML

Marker type	Content	Reference
DITAAlias	Alternate name for FrameMaker format for the current block	15.4.3.6
DITAAttribute	Attributes other than ID of a non- <code><xref></code> block element or parent	15.4.6.4
DITACloseAfter	Ancestor elements to be closed just after current block element ends	15.5.9.2
DITACloseBefore	Ancestor elements to be closed just before current block element starts	15.5.9.1
DITACode	XML code to be inserted at the marker location	
DITAElemID	ID attribute for the current block element	15.4.6.1
DITAEndElem	Tag name for an inline element, to place at the end of the span	15.4.4.4
DITAFirst	Ancestor elements under which the current block element must be first	15.5.6
DITALevel	Level where the current block element should appear in the DITA file	15.5.13
DITALinkElemID	ID of the link target for the next <code><xref></code> element	15.10.3
DITALinkFormat	Format attribute of the next <code><xref></code> element	15.10.6.2
DITALinkScope	Scope attribute of the next <code><xref></code> element	15.10.6.1
DITALinkType	Type attribute of the next <code><xref></code> element	15.10.6.3
DITANavTitle	Alternate title for navigation use	15.9.5
DITANoToc	Adds <code>toc="no"</code> to the <code>topicref</code> to the current topic, suppresses <code>navtitle</code>	15.9.6
DITAOpenAfter	Elements to be opened just after current block element ends	15.5.10.2
DITAOpenBefore	Enclosing elements to be opened just before current block element starts	15.5.10.1
DITAParent	Required parents for the current block element	15.5.2
DITAParentID	ID of the interpolated parent of the current block element	15.4.6.1
DITASearchTitle	Alternate title for search-result use	15.9.5
DITAShortDesc	<code><shortdesc></code> element for the current topic	15.4.9
DITAStartElem	Tag name for an inline element, to place at the start of the span	15.4.4.4

Table 15-3 Predefined marker types for DITA XML (continued)

Marker type	Content	Reference
DITATag	Element name mapping for the current block (not inline) element	15.4.3.5
DITATopic	DTD type for the current topic	15.9.2
DITATopicID	ID attribute for the current topic	15.9.3
DITATopicOutputclass	Outputclass attribute for the root topic	15.4.6.6
DITATopicRootAttrs	Attributes for the root element of a topic	15.4.6.3

16 Configuring DITA maps

Mif2Go creates maps for DITA projects, from both structured and unstructured FrameMaker documents. This section shows how to configure DITA maps and optionally construct a DITA bookmap. Topics include:

- §16.1 [Understanding how Mif2Go generates DITA maps](#) on page 539
- §16.2 [Configuring DITA ditamaps](#) on page 539
- §16.3 [Constructing a DITA bookmap](#) on page 548
- §16.4 [Mapping FrameMaker files to bookmap components](#) on page 551
- §16.5 [Providing attributes for bookmap wrapper elements](#) on page 555
- §16.6 [Overriding DITA map settings with markers](#) on page 556

See also:

- §15 [Converting to DITA XML](#) on page 473
- §32 [Working with content models](#) on page 905

16.1 Understanding how Mif2Go generates DITA maps

Mif2Go generates a DITA map for each FrameMaker file in your document. The map file has the base name of the FrameMaker file from which the map was generated, with extension `.ditamap`. DITA maps are based on configuration settings and on the implied structure of your FrameMaker document. If the first map level number is greater than 1, **Mif2Go** normalizes the remaining levels to be relative to the first.

16.2 Configuring DITA ditamaps

In this section:

- §16.2.1 [Specifying options for ditamaps](#) on page 539
- §16.2.2 [Specifying topic levels in ditamaps](#) on page 544
- §16.2.3 [Accounting for missing topic levels](#) on page 544
- §16.2.4 [Specifying roles for topics in ditamaps](#) on page 545
- §16.2.5 [Adding relationship tables to ditamaps](#) on page 546
- §16.2.6 [Providing navigation aids in ditamaps](#) on page 547

See also:

- §32.7.4 [Overriding declarations in a DITA map content model](#) on page 915

16.2.1 Specifying options for ditamaps

In this section:

- §16.2.1.1 [Choosing whether to overwrite ditamaps](#) on page 540
- §16.2.1.2 [Choosing whether a ditamap references maps or topics](#) on page 540
- §16.2.1.3 [Specifying the base file name for a ditamap](#) on page 541
- §16.2.1.4 [Specifying a title for a chapter or book ditamap](#) on page 541
- §16.2.1.5 [Specifying a navigation title for a ditamap](#) on page 542
- §16.2.1.6 [Specifying the ID for a ditamap](#) on page 542
- §16.2.1.7 [Excluding topic IDs for FrameMaker 8 import](#) on page 543
- §16.2.1.8 [Excluding <topicmeta> elements for FrameMaker 8 import](#) on page 543

16.2.1.1 Choosing whether to overwrite ditamaps

By default, **Mif2Go** rewrites `.ditamap` files each time you run a DITA conversion. Unless the changes you make between conversion runs (changes either to your FrameMaker document or to configuration files) have no effect at all on DITA structure, most likely something will be different in one or more generated maps. On the other hand, if you have edited book or chapter maps by hand, you would not want to lose your edits.

To prevent **Mif2Go** from overwriting existing DITA maps:

```
[DITAOptions]
; WriteDitamaps = Yes (default) overwrite existing maps,
;   or No (when using hand-edited chapter and book-level maps)
WriteDitamaps=No
```

When `WriteDitamaps=Yes` (the default), existing maps are overwritten. Also, if a FrameMaker file is in a book, and the book file is open, **Mif2Go** rebuilds the map for the book whenever you convert the file.

If you change the setting of `WriteDitamaps` from `Yes` to `No` for subsequent conversions of the same project, and you make any changes that affect DITA structure, maps can get out of synchronization with topics.

If you set `WriteDitamaps=No` the *first* time you run a DITA conversion, **Mif2Go** will not produce any DITA maps.

16.2.1.2 Choosing whether a ditamap references maps or topics

Ideally, a chapter map references topics, and a book map references chapter maps. However, not all DITA tools allow nested maps. Therefore the default **Mif2Go** option is to have the book map reference topics directly rather than reference the chapter maps.

In this section:

§16.2.1.2.1 [Configuring a book map to reference chapter maps](#) on page 540

§16.2.1.2.2 [Configuring chapter maps for FrameMaker 8 import](#) on page 540

16.2.1.2.1 Configuring a book map to reference chapter maps

When you convert a FrameMaker book to DITA XML, **Mif2Go** creates a `.ditamap` file for the book. By default, this book map includes a `<topicref>` for each topic in your project. You can have the book map reference each chapter map instead.

To include in the book `.ditamap` references to chapter maps instead of direct references to each topic:

```
[DITAOptions]
; MapBookTopics = Yes (default, include <topicref> for each topic in
;   book .ditamap), or No reference the chapter maps instead)
MapBookTopics = No
```

Referencing chapter maps is better practice, if your downstream tools support this approach. If you plan to re-import DITA files into FrameMaker version 8, there are additional considerations; see §16.2.1.2.2 [Configuring chapter maps for FrameMaker 8 import](#) on page 540.

If you are producing DITA version 1.1 output that includes a `<bookmap>`, also see §16.3.5 [Choosing whether a bookmap references maps or topics](#) on page 550.

16.2.1.2.2 Configuring chapter maps for FrameMaker 8 import

If you expect to import your DITA output into FrameMaker version 8, make sure there is a single top-level topic in each chapter map; then the book-level map will work as expected.

What happens depends on whether your book-level ditamap references other ditamaps or DITA files:

- If your book-level ditamap references other ditamaps that reference DITA files, and you ask FrameMaker 8 to make your book-level map into a FrameMaker book, you will get an error message (“processing instruction ignored”) for each file, but the resulting book, and all its chapters, will be correct.
- If your book-level ditamap references DITA files directly, each DITA file (or each nested set of topicrefs) will become a chapter, which may be desirable for such purposes as creating a library of DITA topics to be referenced elsewhere as insets.

The Leximation DITA-FMx plug-in permits both.

See also:

§15.8.4 [Nesting DITA topics in unsplit files](#) on page 521

§15.8.5 [Wrapping DITA topics in a top-level <dita> element](#) on page 521

§16.2.1.7 [Excluding topic IDs for FrameMaker 8 import](#) on page 543

§16.2.1.8 [Excluding <topicmeta> elements for FrameMaker 8 import](#) on page 543

16.2.1.3 Specifying the base file name for a ditamap

By default, the file name for each DITA map file is the base name of the FrameMaker file from which the map was generated, with extension `.ditamap`. This is true for both book files and chapter files.

To specify a different base name for a chapter map file, include the following setting in a chapter configuration file `filename.ini` named for the FrameMaker file (see §33.1.1 [Providing configuration files for individual chapters](#) on page 919):

```
[DITAOptions]
; MapName = name to use (without extension) for the chapter ditamap.
MapName = othername
```

Alternatively, you can insert a **DITAMapName** marker in the FrameMaker file. The content of the **DITAMapName** marker becomes the base name for the map file, overriding any value specified for MapName.

To specify a different base name for a book map file, in your project configuration file:

```
[DITAOptions]
; BookMapName = name to use (without extension) for the book ditamap.
BookMapName = bookname
```

You cannot use a marker to override the base name for a book map file.

16.2.1.4 Specifying a title for a chapter or book ditamap

For the `<title>` element of a chapter map, by default **Mif2Go** uses the FrameMaker chapter title.

*Arbitrary chapter
map title*

To specify a different title for a chapter map, include the following setting in a chapter-specific configuration file named for the FrameMaker file (see §33.1.1 [Providing configuration files for individual chapters](#) on page 919):

```
[DITAOptions]
; MapTitle = content of <title> element for chapter map
MapTitle = This is the title of the current chapter map
```

When you provide a value for MapTitle, **Mif2Go** sets a `<title>` element at the start of the chapter map (effectively at map level 0); the value assigned to MapTitle becomes the content of the element.

Alternatively, you can insert a **DITAMapTitle** marker in the FrameMaker chapter file. The content of the marker becomes the content of the <title> element for the map generated from that file, overriding any value specified for MapTitle.

First topic title as chapter map title

To use the title of the first topic as the chapter map title:

```
[DITAOptions]
; UseAltMapTitle = No (default) or Yes (if no title specified for map,
; use the navtitle of the first topic in the file as the map title)
UseAltMapTitle = Yes
```

When UseAltMapTitle=Yes, the alternate title overrides the value of MapTitle.

Book map title

To specify a title for the book map that is generated from a FrameMaker book file, in your project configuration file:

```
[DITAOptions]
; BookMapTitle = content of <title> element for book map.
BookMapTitle = This is the title of the book map
```

You cannot use a marker to override the title for a book map file.

16.2.1.5 Specifying a navigation title for a ditamap

To specify a navigation title for a DITA map, include the following setting in a chapter configuration file *filename.ini* named for the FrameMaker file (see §33.1.1 [Providing configuration files for individual chapters](#) on page 919):

```
[DITAOptions]
; MapHead = navigation title for chapter map
MapHead = Title for use by navigation links
```

Alternatively, you can insert a **DITAMapHead** marker in the FrameMaker file. The content of the marker becomes the navtitle attribute for the map generated from the file, overriding any value specified for MapHead.

When you provide a value for MapHead or insert a **DITAMapHead** marker, **Mif2Go** creates a <topichd> element at the start of the chapter map (effectively at map level 0) that contains the rest of the map entries.

16.2.1.6 Specifying the ID for a ditamap

By default, the value of the id attribute for each DITA map is the base name of the FrameMaker file from which the map was generated. This is true for both book files and chapter files.

Mif2Go writes a map for each FrameMaker chapter. If your FrameMaker chapter file names contain characters that are not valid in the id attribute, you can prevent **Mif2Go** from creating map @ids:

```
[DITAOptions]
; UseMapID = Yes (default) or No (omit entirely)
UseMapID = No
```

The id attribute has no use in DITA maps, so is not essential. However, without it you might not know just which chapter the map was made for. The map file name usually tells you, but if you change the file name to fit with a CMS requirement, for example, the @id might be all you have left.

To specify a different map @id for a particular chapter, include the following setting in a chapter-specific configuration file named for the FrameMaker file (see §33.1.1 [Providing configuration files for individual chapters](#) on page 919):

```
[DITAOptions]
; MapID = id for Frame chapter file ditamap, default is base file name
MapID = otherid
```

Alternatively, you can insert a **DITAMapID** marker in the FrameMaker file. The content of the marker becomes the `id` attribute for the map generated from the file, overriding any value specified for `MapID`.

To specify a different book map `@id`, in your project configuration file:

```
[DITAOptions]
; BookMapID = id for book file ditamap, default is base book file name
BookMapID = bookid
```

You cannot use a marker to override the `id` attribute of a map generated from a FrameMaker book file.

16.2.1.7 Excluding topic IDs for FrameMaker 8 import

The setting described in this section applies only if you expect to re-import your DITA output into FrameMaker version 8, without using the Leximation DITA-FMx plug-in.

Without DITA-FMx, FrameMaker version 8 can open topic files referenced from a `<ditamap>` via `TopicFileName.dita#TopicID` if and only if the topics in the file are inside a composite `<dita>` wrapper (see §15.8.5 [Wrapping DITA topics in a top-level <dita> element](#) on page 521). However, the chapter files produced by FrameMaker 8 from those topics have names of the form `TopicFileName#TopicID.fm`, which is probably not what you want. To allow FrameMaker 8 to reconstruct your original FrameMaker chapters, you must omit topic IDs from `href` attributes in maps.

To omit topic IDs from map references to topics:

```
[DITAOptions]
; MapTopicID = Yes (default, include topic ID with filename in maps)
; or No (use filename without ID, for Frame 8 import without
; DITA-FMx).
MapTopicID = No
```

When `MapTopicID=No`, you must split FrameMaker chapters into individual topic files for the ID-less map references to work. See §15.8.2 [Splitting FrameMaker files into DITA topic files](#) on page 520.

Note: When `[DITAOptions]FM8Import=Yes`, **Mif2Go** changes the default value of `MapTopicID` to `No`; see §15.2.6 [Ensuring FrameMaker 8 import compatibility](#) on page 481.

16.2.1.8 Excluding <topicmeta> elements for FrameMaker 8 import

The setting described in this section applies only if you expect to re-import your DITA output into FrameMaker version 8, without using the Leximation DITA-FMx plug-in.

Without DITA-FMx, on import to FrameMaker version 8, if a `<ditamap>` contains any `<topicmeta>` elements, the text content of those elements is tacked onto the end of the topic file, after the closing tag of the root element. This causes an error message, and the text turns up at the end of the FrameMaker topic file. The only remedy is to eliminate `<topicmeta>` elements entirely.

To exclude `<topicmeta>` elements from map `<topicref>` elements:

```
[DITAOptions]
; MapTopicmeta = Yes (default, include topicmeta with
; searchtitle and shortdesc, if any, in map topicrefs)
; or No (omit topicmeta, required for Frame 8 import
```

```
; without DITA-FMx).
MapTopicmeta = No
```

Note: When [DITAOptions]FM8Import=Yes, **Mif2Go** changes the default value of MapTopicmeta to No; see §15.2.6 [Ensuring FrameMaker 8 import compatibility](#) on page 481.

16.2.2 Specifying topic levels in ditamaps

To specify map levels for topics, assign a level number to each FrameMaker format that you map to a <title> element at the topic level in [DITAParaTags]; see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487. For example:

```
[DITAMapLevels]
; Frame para format = level of topics it starts in ditamap, default 1.
Heading1 = 1
Heading2 = 2
```

Each instance of a paragraph format assigned a level number generates a <topicref> that nests the lower <topicref> elements. For paragraph formats that are not listed here, but that are mapped to <title> elements at the topic level in [DITAParaTags], **Mif2Go** uses any level assignments you have provided for those formats in [HelpContentsLevels]. See §7.4.4 [Setting contents levels for HTML-based Help](#) on page 210.

If you direct **Mif2Go** to nest topics (see §15.8.4 [Nesting DITA topics in unsplit files](#) on page 521), **Mif2Go** adjusts map levels as follows:

- If a topic is nested in a glossary topic (which is invalid), the map level of the nested topic is decreased to make it a sibling of the glossary topic.
- If a topic is set to a level more than one deeper than the previous topic, its level is decreased so that no levels are skipped.

Any adjustments to map levels also affect the chapter ditamap.

See also:

§15.5.13 [Specifying DITA element levels](#) on page 509

16.2.3 Accounting for missing topic levels

If your FrameMaker document uses a hierarchy of heading formats, such as:

```
Heading1
  Heading2
    Heading3
      Heading4
```

And you have specified the following map levels for these headings (see §16.2.2 [Specifying topic levels in ditamaps](#) on page 544):

```
[DITAMapLevels]
Heading1 = 1
Heading2 = 2
Heading3 = 3
Heading4 = 4
```

But at least some of the time a heading is skipped in FrameMaker, such as:

```
Heading1
  Heading2
    Heading4
```

Wherever a *Heading4* follows a *Heading2*, **Mif2Go** promotes that *Heading4* to map level 3, because DITA does not allow skipped levels in a map. Additional instances of *Heading4* that follow will remain at map level 4, and thus appear to be subordinate to the instance that was promoted to map level 3. How you adjust for this problem depends on whether a skipped level tends to be the rule or the exception in your FrameMaker document:

[Heading3 present in most files](#)

[Heading3 absent from most files](#)

[Heading3 occasionally absent in a file.](#)

*Heading3 present
in most files*

If *Heading3* is consistently present in most files and consistently missing in some, you would assign map level 4 to *Heading4* in your project configuration file; and for each FrameMaker file where *Heading3* is missing, include in the project directory a chapter-specific *filename.ini* that contains the following setting:

```
[DITAMapLevels]
Heading4 = 3
```

For example, if *intro.fm* has no instances of *Heading3*, include this setting in *intro.ini*. **Mif2Go** will use this setting as an override, for *intro.fm* only. See §33.1.1 [Providing configuration files for individual chapters](#) on page 919.

*Heading3 absent
from most files*

If *Heading3* is consistently missing from most files and consistently present in some, you would assign map level 3 to *Heading4* in your project configuration file; and for the files that use *Heading3*, include in the project directory a chapter-specific *filename.ini* for each that contains the following settings:

```
[DITAMapLevels]
Heading3 = 3
Heading4 = 4
```

The setting for *Heading4* will override the same setting in the project configuration file.

*Heading3
occasionally
absent in a file*

If *Heading3* is only occasionally missing in a file where it is usually present, you would have to use a **Config** marker to set the map level for *Heading4* to 3 just before a group of *Heading4*s that are not subordinate to a *Heading3*, and another **Config** marker after the group to change the map level back to 4. See §33.2 [Overriding settings with markers or macros](#) on page 920.

16.2.4 Specifying roles for topics in ditamaps

To specify the kind of entry (if any) each topic type should have in its chapter map:

```
[DITAMapUsage]
; Frame para format that starts topic = Topic (default), Head, or None
ParaFmt=Topic
```

You can assign one of the values *Topic*, *Head*, or *None* to each paragraph format that you map to a *<title>* element at the topic level in *[DITAParaTags]*; see §15.4.3 [Mapping paragraph formats to DITA block elements](#) on page 487. These values have the following effects:

<i>Topic</i>	(<i>Default</i>) Include a <i><topicref></i> element in the map for this topic
<i>Head</i>	Include a <i><topichead></i> element only, with a title but no link
<i>None</i>	Exclude the topic from the map

Because the default value is *Topic*, you need list only those paragraph formats that identify topics that should not be linked from the map, or that should be excluded from the map.

See also:

§15.4.9 [Providing a *<shortdesc>* element for a DITA topic](#) on page 500

16.2.5 Adding relationship tables to ditamaps

Mif2Go creates a relationship table for each chapter map generated from a FrameMaker file.

In this section:

§16.2.5.1 [Understanding how Mif2Go creates relationship tables](#) on page 546

§16.2.5.2 [Excluding the ALink column from relationship tables](#) on page 546

§16.2.5.3 [Adding ALink rows to relationship tables](#) on page 546

§16.2.5.4 [Specifying one-way linking for a topic in a relationship table](#) on page 547

§16.2.5.5 [Specifying a collection-type attribute for each topic type](#) on page 547

16.2.5.1 Understanding how Mif2Go creates relationship tables

By default, **Mif2Go** creates relationship tables with four columns: the first column contains ALink subject names that apply to cells in the same row in the other columns. The ALink column is followed by the usual three columns for topic types concept, task, and reference.

Each ALink subject name in the first column is expressed in a <data> element:

```
<data name="subject" value="ALink term"/>
```

Each row in a default relationship table has <topicref> elements for all topics that include the ALink term named in the first column, sorted into cells by topic type.

16.2.5.2 Excluding the ALink column from relationship tables

If the DITA tools you use cannot accommodate relationship tables that include the extra ALink column, you can instruct **Mif2Go** to omit that column.

To prevent **Mif2Go** from including an ALink column in relationship tables:

```
[DITAOptions]
; UseRelNameColumn = Yes (default, first col of reltable has ALink
; name in <data> element) or No (use only usual type columns).
UseRelNameColumn = No
```

If your FrameMaker document does not include ALink references, and you do not insert any **DITARelRow** marker (see §16.2.5.3 [Adding ALink rows to relationship tables](#) on page 546), **Mif2Go** does not produce a relationship table for the map.

16.2.5.3 Adding ALink rows to relationship tables

To add a subject row to a relationship table, in FrameMaker insert a **DITARelRow** marker in the topic. The content of a **DITARelRow** marker is a subject name, which becomes the name of a row in the table. Each row in the table corresponds to one subject name, so topics with the same subject name all appear in the same row.

The same topic can appear in multiple rows, sharing each row with all other topics marked with the same subject name. Each subject name generates its own row, even if that row is otherwise identical to other generated rows.

DITARelRow markers are equivalent to **ALink** markers in Help systems. If your document already contains ALink markers, you can clone them to create **DITARelRow** markers.

To clone ALink markers for relationship-table entries, include the following setting in your project configuration file:

```
[MarkerTypes]
ALink=DITARelRow
```

16.2.5.4 Specifying one-way linking for a topic in a relationship table

To specify a value for the linking attribute of a `<topicref>` in a relationship table, insert a **DITARelLinking** marker in the topic in FrameMaker. The content of a **DITARelLinking** marker can be one of the following:

```
targetonly
sourceonly
```

16.2.5.5 Specifying a collection-type attribute for each topic type

To specify a collection-type attribute for a topic type in a **Mif2Go**-generated relationship table for a chapter map:

```
[DITARelGroups]
; DITA type name = collection-type attribute to use in the <colspec>
; for the chapter map <reltable> column for that DITA type.
topictype = colltype
```

To specify a collection-type attribute for a topic type in a relationship table for a book map:

```
[DITARelBookGroups]
; DITA type name = collection-type attribute to use in the <colspec>
; for the book map <reltable> column for that DITA type.
topictype = colltype
```

If you do not include a setting for a topic type in `[DITARelBookGroups]`, **Mif2Go** uses the setting for that topic type (if any) in `[DITARelGroups]` for the relationship table for the book map.

We suggest the following collection-type attributes; however, you can specify others:

```
family      (equivalent to the usual ALink behavior)
sequence    (links are in order of appearance in the chapter)
choice      (processor dependent)
```

The default is no collection-type, so that topics in that column do not link to each other, but only to topics in other columns in the same row.

For example:

```
concept = family
task = sequence
```

These settings would yield the following results:

- Concept topics would link to other concepts, and to task and reference topics in the same row (same **DITARelRow** value; see §16.2.5.3 [Adding ALink rows to relationship tables](#) on page 546).
- Task topics would link to each other as an ordered set, and to concept and reference topics normally.
- Reference topics would link to tasks and concepts in the same row, but not to each other.

16.2.6 Providing navigation aids in ditamaps

To add navigation elements to the map entry for a topic, insert custom FrameMaker markers in the `<title>` paragraph of the topic. [Table 16-1](#) shows the content and placement of the navigation element for each custom marker type.

Table 16-1 DITA map navigation elements from custom markers

Custom marker type	Marker content	Map element	Map placement
DITAAncor	id attribute	<anchor>	After the <topicref> of the topic
DITANavref	mapref attribute	<navref>	After the <topicref> of the topic
DITAMapref	href attribute	<topicref>	After the <topicref> of the topic
DITALinkText	text for the link	<linktext>	In the map <topicmeta>

The content of a **DITAMapref** marker should be a map file name. **Mif2Go** sets the format attribute of the resulting <topicref> element to "ditamap".

16.3 Constructing a DITA bookmap

Much of the information required for a DITA bookmap is not present in FrameMaker files. This information consists of metadata that you specify in configuration files and macro files. **Mif2Go** supports bookmaps for DITA version 1.1.

In this section:

- §16.3.1 [Specifying the type of map for a book](#) on page 548
- §16.3.2 [Specifying <booktitle> information](#) on page 548
- §16.3.3 [Specifying <bookmeta> information](#) on page 549
- §16.3.4 [Extending <part> to include <appendix>](#) on page 550
- §16.3.5 [Choosing whether a bookmap references maps or topics](#) on page 550
- §16.3.6 [Excluding the book-level reltable from a bookmap](#) on page 550

See also:

- §32.7.4 [Overriding declarations in a DITA map content model](#) on page 915

16.3.1 Specifying the type of map for a book

To specify the type of map to include for a FrameMaker book:

```
[DITABookmapOptions]
; BookmapType = ditamap (default) or bookmap (1.1 only)
BookmapType = bookmap
```

When BookmapType=ditamap, **Mif2Go** uses settings described in §16.2 [Configuring DITA ditamaps](#) on page 539.

When BookmapType=bookmap, **Mif2Go** uses the values of settings in section [DITABookmapFiles] to categorize bookmap components; see: §16.3 [Constructing a DITA bookmap](#) on page 548.

16.3.2 Specifying <booktitle> information

The following settings provide values that comprise the bookmap <booktitle> element:

```
[DITABookMapOptions]
; BookTitle = <mainbooktitle> for bookmap
BookTitle = My Book Title
; BookSubtitle = optional <booktitlealt> for bookmap
BookSubtitle = My Subtitle for the Book
; BookLibrary = optional library catalog information text, can be a
; macro file or [section] reference.
BookLibrary = <$/mydescription.txt>
```

If you do not include a setting for `BookTitle`, **Mif2Go** uses instead the value of `[DITAOptions]BookMapTitle`, in a `<title>` element; see §16.2.1.4 [Specifying a title for a chapter or book ditamap](#) on page 541.

You can optionally provide content for the `<booklibrary>` element, in either of the following ways:

- a separate macro file; for example:

```
[DITABookMapOptions]
BookLibrary = $./path/to/booklib.txt
```
- a macro section in the same configuration file; for example:

```
[DITABookMapOptions]
BookLibrary = <$LibText>

[LibText]
Text description ...
```

See §28.1 [Defining and invoking macros](#) on page 787.

16.3.3 Specifying `<bookmeta>` information

You must provide valid XML content for the `<bookmeta>` element. Most of the information cannot be derived from your FrameMaker document. An annotated template for the `<bookmeta>` element, `bookmeta.xml`, is available in your **Mif2Go** distribution directory; the contents of this template are listed in §E [DITA `<bookmeta>` template](#) on page 1039.

An easy way to supply `<bookmeta>` content is to do the following:

1. Copy `bookmeta.xml` from your **Mif2Go** distribution directory to the project directory for your DITA project (or to another directory), and optionally rename the copy.
2. Delete from the copy elements you do not need; also delete any non-XML text.
3. Substitute appropriate values for the elements you do need, following XML rules.
4. Optionally validate the XML against the DITA version 1.1 standard.
5. Copy the entire content into your project configuration file as a macro section, or reference the template copy as a macro file.

To specify the location of your `<bookmeta>` information:

```
[DITABookMapOptions]
; BookMeta = DITA content, can be macro file or [section]
BookMeta = <$.\mymetadata.xml>
```

If you do not include a setting for `BookMeta`, the `<bookmeta>` element is omitted from output.

You can reference `<bookmeta>` content in either of the following ways:

- a separate macro file; for example:

```
[DITABookMapOptions]
BookMeta = <$.\path\to\bookmeta.xml>
```
- a macro section in your project configuration file; for example:

```
[DITABookMapOptions]
BookMeta = <$BookmetaElements>

[BookmetaElements]
<bookmeta>
. . .
</bookmeta>
```

See §28.1 [Defining and invoking macros](#) on page 787.

16.3.4 Extending <part> to include <appendix>

To comply with the DITA 1.1 specification, by default **Mif2Go** does not allow <appendix> elements within <part> in a bookmap. However, **Mif2Go** makes it possible to go beyond the specification and include <appendix> as well as <chapter> in <part>.

To include <appendix> elements in a bookmap <part> element:

```
[DITABookmapOptions]
; AllowPartAppendix = No (default, per DITA spec)
; or Yes (allow a part element in bookmap to include appendix
AllowPartAppendix = Yes
```

When AllowPartAppendix=Yes, you can assign dividers to appendix files as well as to chapter files; see §16.4.4 [Assigning a divider role to a section file or chapter](#) on page 554.

Note: A bookmap with <appendix> inside <part> is not valid DITA 1.1.

16.3.5 Choosing whether a bookmap references maps or topics

When a FrameMaker file has a starting topic whose topicref wraps the rest of the topics in that file, **Mif2Go** replaces the topicref tag with the appropriate derivative, such as chapter. Where there is no single top-level topic, **Mif2Go** wraps the whole lot in a new top-level tag with no href or navtitle. You can provide values for these tags; see §16.5 [Providing attributes for bookmap wrapper elements](#) on page 555.

The following setting determines whether a bookmap references chapter ditamaps or references topics directly:

```
[DITAOptions]
; MapBookTopics = Yes (default, include <topicref> for each topic in
; book .ditamap), or No reference the chapter maps instead)
MapBookTopics = No
```

If MapBookTopics=No, the href attribute in a bookmap topicref is to the ditamap generated for the FrameMaker chapter file.

If MapBookTopics=Yes, and the chapter file has one top-level topic in its map, that topicref is renamed as the chapter (or other appropriate) element. If the chapter file has more than one top-level topicref, its topicrefs are instead wrapped in the <chapter> element, which in that case has no href attribute unless you supply one in [DITABookmapHrefs]; see §16.5 [Providing attributes for bookmap wrapper elements](#) on page 555.

See also:

§16.2.1.2 [Choosing whether a ditamap references maps or topics](#) on page 540

16.3.6 Excluding the book-level reltable from a bookmap

By default, **Mif2Go** includes a reltable for the bookmap itself when you generate DITA output from a FrameMaker book. You can choose to omit this reltable:

```
[DITAOptions]
; MapBookRelTable = Yes (default, include reltable in book-level map
; when MapBookTopics=Yes), or No (always exclude reltable from book-
; level map)
MapBookRelTable = No
```

See also:

§16.3.5 [Choosing whether a bookmap references maps or topics](#) on page 550

16.4 Mapping FrameMaker files to bookmap components

Mif2Go does a reasonable job of constructing a <bookmap> for simple cases where your FrameMaker book consists of chapters, TOC, IX, other generated files in front or in back, and perhaps non-generated files before the TOC, such as a cover page. You can run the conversion first without any files mapped to <bookmap> components, then add settings as needed.

In this section:

§16.4.1 [Assigning bookmap roles to FrameMaker files](#) on page 551

§16.4.2 [Assigning frontmatter and backmatter roles and components](#) on page 552

§16.4.3 [Including multiple booklist components of the same type](#) on page 553

§16.4.4 [Assigning a divider role to a section file or chapter](#) on page 554

§16.4.5 [Assigning a series of roles to a single FrameMaker file](#) on page 554

§16.4.6 [Assigning a single role to a series of FrameMaker files](#) on page 554

§16.4.7 [Including placeholders for additional bookmap elements](#) on page 555

16.4.1 Assigning bookmap roles to FrameMaker files

To indicate the role and (if relevant) component that each file in your FrameMaker book should play in the <bookmap>:

```
[DITABookmapFiles]
; filename without .ext = role, optionally followed by a component
; or by Lists then a booklist component, optionally followed by
; ElementBefore or ElementAfter; or None (omit from bookmap).
```

Table 16-2 lists the roles you can assign to FrameMaker files. The role (and components, if any) assigned to a FrameMaker file determine which element or elements are included in the bookmap for the DITA topic or topics generated from that file. Entries appear in the <bookmap> element in the same order they appear in your FrameMaker book. If you need a different sequence, create a different FrameMaker .book file and populate it with files in the sequence you need; then construct the DITA <bookmap> from that alternate FrameMaker book.

Table 16-2 Roles of component files in a bookmap

Role	Book component	Ref.
Front	Frontmatter, including one or more components; place first after =	16.4.2
Back	Backmatter, including one or more components; place first after =	16.4.2
Lists	Generated file; must follow Front or Back.	16.4.2
Part	Divider for logical groups of chapters, either explicit or implicit	16.4.4
Chapter	Normal chapter files	
Appendix	Normal appendix files	
*Seg	Multiple-role chapters	16.4.5
	Multiple-chapter roles	16.4.6
ElementBefore	Elements to be generated precede this file	16.4.7
ElementAfter	Elements to be generated follow this file	16.4.7
None	Omit from the bookmap	

<i>Front, Back, and Lists</i>	For files that belong in frontmatter or backmatter, Front or Back must precede any other role or component. If you assign Front or Back to a file, that role applies to the following component, if any. Lists must follow Front or Back, and must be followed by a booklist component. See §16.4.2 Assigning frontmatter and backmatter roles and components on page 552.
<i>Authored vs. generated files</i>	Most of the non-generated files in your FrameMaker book should be assigned either the Chapter or Appendix role. Because Chapter is the default role, you do not have to list single-role chapter files in [DITABookmapFiles], except for chapters that begin implicit multi-chapter sections; see §16.4.4 Assigning a divider role to a section file or chapter on page 554.
<i>Omitting a file from the list</i>	If you do not list in [DITABookmapFiles] a particular file in your FrameMaker book, Mif2Go includes that file in the bookmap based on file type (authored vs. generated) and on the position of the file relative to files that are listed. A non-generated file that occurs after the TOC (if any) becomes a <chapter> element.
<i>Omitting all files from the list</i>	If you do not list any files, the bookmap contains one entry for each FrameMaker file, with a <frontmatter> element for the FrameMaker TOC and a <backmatter> element for the FrameMaker IX. References to any other generated files appear in <frontmatter> or <backmatter> as <booklist> elements, based on their location in the FrameMaker book. All non-generated files become <chapter> elements, except for any preceding the TOC (such as a cover page), which become <topicref> elements in <frontmatter>.
<i>Omitting a file from the bookmap</i>	When <i>filename</i> =None, the file listed is not included in the bookmap.

16.4.2 Assigning frontmatter and backmatter roles and components

When *filename*=Front, follow Front with one of Lists (default), Notice, Dedication, Colophon, Abstract, Draftintro, or Preface. For example:

```
[DITABookmapFiles]
legalpage = Front Notice
```

When *filename*=Back, follow Back with one of Lists (default), Notice, Dedication, Colophon, or Amendment. For example:

```
[DITABookmapFiles]
endpage = Back Colophon
```

When *filename*=Front Lists or *filename*=Back Lists (or just Front or Back), follow Front, Back, or Lists with one of TOC (default for Front Lists), LOF, LOT, Abbr, Trademark, Biblio, Glossary, IX (default for Back Lists), or Booklist (used for other types of FrameMaker generated files such as LOM and IOM). For example:

```
[DITABookmapFiles]
mybookTOC = Front Lists TOC
mybookLOF = Front Lists LOF
keywords = Back Lists Booklist
mybookIX = Back Lists IX
```

[Table 16-3](#) shows the roles each component can follow in frontmatter or backmatter.

Table 16-3 Components for bookmap frontmatter and backmatter

Component	DITA element	Can follow these roles in [DITABookmapFiles]:			
		Front	Back	Front Lists	Back Lists
Lists	<booklists>	Yes	Yes		
Notice	<notices>	Yes	Yes		
Dedication	<dedication>	Yes	Yes		
Colophon	<colophon>	Yes	Yes		
Abstract	<bookabstract>	Yes			
Draftintro	<draftintro>	Yes			
Preface	<preface>	Yes			
Amendment	<amendments>		Yes		
TOC	<toc>			Yes	Yes
LOF	<figurelist>			Yes	Yes
LOT	<tablelist>			Yes	Yes
Abbr	<abbrevlist>			Yes	Yes
Trademark	<trademarklist>			Yes	Yes
Biblio	<bibliolist>			Yes	Yes
Glossary	<glossarylist>			Yes	Yes
IX	<indexlist>			Yes	Yes
Booklist	<booklist>			Yes	Yes

Booklist components

TOC, and all components that have an element name that ends in `list`, are booklist items and must follow `Lists`, which should follow non-booklist items. Booklist items should be together, at one end or the other of `Front` or `Back`, preferably following any other items. However, you might have to put some booklist items before other items; for example, to get the glossary and index before the colophon:

```
[DITABookmapFiles]
endpage = Back Lists Glossary IX Colophon
```

If you omit `Lists` before a booklist item, or `Front` or `Back` before `Lists`, **Mif2Go** can usually figure out what is needed and interpolate the missing item. However, it is best to be explicit about which items belong where.

16.4.3 Including multiple booklist components of the same type

If you need more than one instance of a component (for example, a second TOC or multiple indexes), append a different number to each instance of the component name:

```
[DITABookmapFiles]
endpage = Back Lists IX1 IX2 IX3
```

In the case of multiple indexes, all FrameMaker markers used to generate all of them must be mapped to `Index` first. This is because all must become `<indexterm>` elements in DITA output, even though they are destined for different `<indexlist>` elements.

To map other index markers to `Index` (for example):

```
[Markers]
RTFkeyword = Index
HTMkeyword = Index
XMLkeyword = Index
```

See §29.3.1 [Remapping and cloning marker types](#) on page 836.

To associate each set of index markers with the corresponding index, assign each set a different @outputclass, and assign the same @outputclass to the IX component where they belong. For example:

```
[IndexMarkerOutputClass]
RTFkeyword = rtfix
HTMkeyword = htmix
XMLkeyword = htmix

[DITABookmapOutputclasses]
IX1 = rtfix
IX2 = htmix
```

See §16.5 [Providing attributes for bookmap wrapper elements](#) on page 555.

You can leave any original FrameMaker **Index** markers and their corresponding IX component (in this example, IX3) without an @outputclass.

16.4.4 Assigning a divider role to a section file or chapter

When *filename*=Part, *filename* logically contains all Chapters until the next Part, or the first Appendix, or the first Back item. Part may precede Chapter if the intended divider is not a file of its own; for example:

```
[DITABookmapFiles]
advanced = Part Chapter
```

Part may precede Appendix when AllowPartAppendix=Yes (see §16.3.4 [Extending <part> to include <appendix>](#) on page 550), although this is not valid DITA 1.1.

16.4.5 Assigning a series of roles to a single FrameMaker file

If a FrameMaker file contains more than one bookmap component, assign role MultiSeg to that file, along with a sequential list of the topicrefs to the starting topics of all components. Also provide a separate entry for each topicref, assigning the roles described in §16.4.1 [Assigning bookmap roles to FrameMaker files](#) on page 551. Use the following syntax:

```
[DITABookmapFiles]
filename = MultiSeg #topicid1 #topicid2 ... #topicidN
#topicid1 = role1 role2 ...
...
#topicidN = role1 role2 ...
```

Each topicref for the specified topic ID includes any topicrefs nested within it. Each topicref listed may be nested at any level in its chapter ditamap, or not nested. List topics in the sequence used in the bookmap; the original sequence in the chapter ditamap is ignored. Any topicref that is not mentioned after MultiSeg, and is not nested in a topicref that is mentioned, is excluded from the bookmap.

16.4.6 Assigning a single role to a series of FrameMaker files

If multiple consecutive FrameMaker files comprise a single bookmap component, use the following syntax to assign the same role to those files:

```
[DITABookmapFiles]
filename1 = StartSeg role component ...
filename2 = ContinueSeg
...
filenameN = EndSeg
```


All `topicrefs` from all the segment chapter ditamaps are wrapped within one new chapter-level element with no `href` attribute.

16.4.7 Including placeholders for additional bookmap elements

You can include placeholders for DITA elements that are not being converted from FrameMaker, but instead will be produced when the bookmap is rendered for presentation; for example, a glossary or an index.

Items you can use for placeholders include roles `Front`, `Back`, and `Lists`, and any of the components in [Table 16-3](#). However, you cannot assign roles `Part`, `Chapter`, or `Appendix`, or a `Seg` directive.

To position a placeholder in the bookmap:

- In `[DITABookmapFiles]`, assign property `ElementBefore` or `ElementAfter` to the file the placeholder should precede or follow.
- In `[BookmapElementBefore]` or `[BookmapElementAfter]`, assign the desired placeholder component to the file.

For example, to specify that one or more DITA elements should precede the preface for the **Mif2Go User's Guide**, FrameMaker file `about.fm`:

```
[DITABookmapFiles]
about = Front Preface ElementBefore
```

To list components for the DITA elements to precede the preface:

```
[BookmapElementBefore]
about = Front Lists TOC LOF LOT
```

And to specify that three different indexes should follow the last appendix file, `m2cmod.fm`:

```
[DITABookmapFiles]
m2cmod = Appendix ElementAfter

[BookmapElementAfter]
m2cmod = Back Lists IX1 IX2 IX3
```

Each component you assign creates an element in the bookmap. [Table 16-3](#) on page 553 shows the name of the element inserted for each component.

16.5 Providing attributes for bookmap wrapper elements

For situations in bookmap construction where **Mif2Go** must provide a wrapper element that does not directly correspond to a file in your FrameMaker book or to a component specified in `[DITABookmapFiles]`, unless you supply values for `navtitle`, `href`, `type`, `format`, `scope`, and `outputclass` attributes, these attributes are not present in the wrapper elements.

Mif2Go uses the settings described in this section only when wrappers are produced for a bookmap. In particular:

- For a FrameMaker file mapped to `Part` as well as to `Chapter`, the `navtitle` attribute applies to the `Part` tag.
- If the file is `MultiSeg`, and a wrapper is needed for one segment, the `#topicID` is used as the file name.
- For a `StartSeg/EndSeg` group, the `StartSeg` file name specifies the `navtitle` for the full wrapper.

- The href attribute is to a topic that is not already in your FrameMaker .book, and is provided as an introduction to the wrapped topics.

To specify a value for the navtitle attribute:

```
[DITABookmapTitles]
; filename (without .ext) or component name = navtitle attribute
; for its wrapper element
```

To specify a value for the outputclass attribute:

```
[DITABookmapOutputclasses]
; filename (without .ext) or component name = outputclass attribute
; for its wrapper element
```

To specify a value for the href attribute:

```
[DITABookmapHrefs]
; filename (without .ext) or component name = href attribute
; for its wrapper element
```

To specify a value for the type attribute:

```
[DITABookmapHrefTypes]
; filename (without .ext) or component name = type attribute
; for its wrapper element, for the specified href.
```

To specify a value for the format attribute:

```
[DITABookmapHrefFormats]
; filename (without .ext) or component name = format attribute
; for its wrapper element, for the specified href.
```

To specify a value for the scope attribute:

```
[DITABookmapHrefScopes]
; filename (without .ext) or component name = scope attribute
; for its wrapper element, for the specified href.
```

16.6 Overriding DITA map settings with markers

You might need to insert markers to override configuration settings for particular DITA maps or for a bookmap. **Mif2Go** provides predefined marker types for this purpose, listed in [Table 16-4](#). Most of these marker types are intended to provide ways to cope with unusual situations.

Table 16-4 Predefined marker types for DITA maps and bookmaps

Marker type	Content	Ref.
DITAAncor	ID attribute of a map <anchor> element	16.2.6
DITALinkText	Text of a map <linktext> element	16.2.6
DITAMapHead	Navigation title for the current chapter map	16.2.1.5
DITAMapID	ID attribute for the current chapter map	16.2.1.6
DITAMapName	Base name (without extension) of map file for current chapter	16.2.1.3
DITAMapref	href attribute of a map <topicref> element	16.2.6
DITAMapTitle	Text of <title> element inserted at map level 0	16.2.1.4
DITANavref	mapref attribute of a map <navref> element	16.2.6
DITARelLinking	Linking attribute for a topic in a relationship table	16.2.5.4
DITARelRow	ALink subject name for a row in a relationship table	16.2.5.3

17 Converting to DocBook XML

Mif2Go generates DocBook projects from both structured and unstructured FrameMaker documents, producing DocBook XML output. This section shows how to configure DocBook-specific options. Topics include:

- §17.1 [Generating DocBook XML with Mif2Go](#) on page 557
- §17.2 [Setting up a DocBook XML project](#) on page 559
- §17.3 [Specifying general options for DocBook](#) on page 562
- §17.4 [Configuring DocBook elements](#) on page 564
- §17.5 [Nesting DocBook block elements](#) on page 573
- §17.6 [Designating ancestors for table elements](#) on page 580
- §17.7 [Specifying options for figure elements](#) on page 581
- §17.8 [Overriding DocBook settings with markers](#) on page 582

See also:

- §32 [Working with content models](#) on page 905

17.1 Generating DocBook XML with Mif2Go

Before you set up a **Mif2Go** DocBook project, be clear about what level of familiarity with DocBook you need, what you intend to do with the output, and what role you want **Mif2Go** to play in producing DocBook output.

In this section:

- §17.1.1 [Understanding what you need to know about DocBook](#) on page 557
- §17.1.2 [Clarifying your purpose for creating DocBook output](#) on page 557
- §17.1.3 [Understanding what information you must supply](#) on page 558

17.1.1 Understanding what you need to know about DocBook

To use **Mif2Go** effectively to produce DocBook output, you need a basic knowledge of DocBook, from study of other materials. Teaching our customers DocBook is beyond the scope of the **Mif2Go User's Guide**. You have to know *what* you want; then perhaps we can tell you *how* to make it happen with **Mif2Go**.

If you are not familiar with DocBook, here is a good starting point:

<http://www.docbook.org/tdg/en/html/docbook.html>

If you intend to produce output from the DocBook XML files **Mif2Go** produces, you will also need *DocBook XSL: The Complete Guide* by Bob Stayton:

<http://sagehill.net/docbookxsl/index.html>

For a reference to DocBook style sheets, see:

<http://docbook.sourceforge.net/release/xsl/current/doc/index.html>

Be aware that conversions to another source format, such as DocBook XML, can be difficult. There are no shortcuts. You might need days or weeks to get it right, working with small test documents, before you can go into production.

17.1.2 Clarifying your purpose for creating DocBook output

Mif2Go supports two general purposes for creating DocBook output from FrameMaker:

Migrate legacy content to DocBook XML

Export current content to DocBook as needed.

A third potential purpose might be to use DocBook as an intermediate step in converting documents from unstructured to structured FrameMaker. You could use **Mif2Go** to produce DocBook XML from your unstructured files, then bring the results back into structured FrameMaker. This should be a lot faster than developing FrameMaker conversion tables.

Migrate legacy content to DocBook XML

When you migrate legacy content from FrameMaker to DocBook XML, completeness is less important than it would be if you retain source in FrameMaker. After converting your document you edit in an XML environment. Even validity can be relaxed, if your existing document does not quite measure up. As long as the XML is well formed, you can use XSLT to make adjustments. You can even run XSLT from within **Mif2Go**. See §34.4 [Executing operating-system commands](#) on page 937.

Export current content to DocBook as needed

To continue using FrameMaker as source, and export content to DocBook as needed, you must interpolate into the DocBook output any data required by DocBook but not needed in FrameMaker. You can use FrameMaker markers or dedicated conditional paragraph formats for file-specific data, and **Mif2Go** configuration settings for general data items such as book revision level. You do not need XSLT for this purpose. In fact, you should not need XSLT at all, unless your FrameMaker document does not follow the same sequence of items that DocBook expects.

With **Mif2Go** you can continue to write in FrameMaker, and get a matching DocBook set any time you need one. And you can produce DocBook output from unstructured as well as from structured FrameMaker.

17.1.3 Understanding what information you must supply

You do not have to use structured FrameMaker to produce DocBook XML with **Mif2Go**. You *can* use structured FrameMaker, provided you use named formats rather than the Word-style formatting some structured-FrameMaker users prefer. Whether you use structured or unstructured FrameMaker, you must arrange the content of your document to fit the DocBook architecture before you can convert to completely valid DocBook output.

Mif2Go does not try to validate the output; you must use a validating parser to check output validity. However, **Mif2Go** does ensure valid parental relationships and first-child restrictions. Valid sequence of items within those constraints has to come from the implied or explicit structure of the FrameMaker document.

Mif2Go support for DocBook requires you to supply the following kinds of information in addition to your FrameMaker document:

[DTD properties](#)

[FrameMaker mappings](#)

[Disambiguation](#)

DTD properties

Mif2Go provides two built-in configurations for content models for DocBook version 4.5: one for articles, and one for books.

If you need to modify one of these content models, you can download a copy from Omni Systems; see §32.2.1 [Obtaining a copy of a built-in content-model](#) on page 906. However, the only valid purpose for modifying a built-in content model would be to correct settings for element types. See §32.6 [Inspecting and correcting element types](#) on page 912.

To replace a content model, use free command-line utility **dtdd2ini** to generate a content model from another DTD, and produce a content-model configuration file for your DocBook project. See §32.2.2 [Generating a content model from a DTD](#) on page 906.

FrameMaker mappings

You must map FrameMaker file information, such as formats, to DocBook elements. This information goes into configuration file `m2docbook.ini`, and possibly into chapter-specific configuration files. You might have to use marker in your FrameMaker document to provide information such as topic IDs, element names, and attributes, in cases where these items cannot be derived from the document.

Note: The name of your FrameMaker book must not duplicate the name of any chapter file.

Disambiguation

In an unstructured FrameMaker document, presentation might be the same for several different usages. **Mif2Go** cannot necessarily determine whether (for example) text tagged `<Italic>` is a computer term, a foreign language term, or a long quote, all of which have different representations in DocBook, based on the context. The onus is on the author to disambiguate these usages, if necessary by inserting DocBook-specific custom markers in individual instances of particular formats. **Mif2Go** does handle a few presentational features automatically; for example, by default forced returns (FrameMaker **Shift+Enter**) are converted to spaces.

17.2 Setting up a DocBook XML project

When you set up a DocBook XML project from within FrameMaker, if configuration file `_m2docbook.ini` is not already present in the project directory, **Mif2Go** creates this configuration file for you; see §3 [Converting a book or document](#) on page 77.

To add or change any of the options described in this section, edit configuration file `_m2docbook.ini`, located in the project directory. Or, to apply the changes to all of your DocBook XML projects, edit the configuration template referenced by `_m2docbook.ini`:

```
%omsyshome%\m2g\local\config\local_m2docbook_config.ini.
```

See §30.5 [Deciding which configuration file to edit](#) on page 856.

In this section:

§17.2.1 [Creating a DocBook project](#) on page 559

§17.2.2 [Choosing set-up options for a DocBook project](#) on page 560

§17.2.3 [Specifying DocBook output options](#) on page 561

17.2.1 Creating a DocBook project

To create a DocBook XML project:

1. Create a directory for DocBook output, separate from the directory where your FrameMaker document is located.
2. With your FrameMaker book or document file open, choose **File > Set Up Mif2Go Export**; the *Choose Project* dialog opens (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).
3. Name your DocBook project, and browse to the project directory you created in [Step 1](#).
4. Choose output type DocBook and click **OK**.

5. Check options in the *Set Up DocBook Project* dialog (see §17.2.2 [Choosing set-up options for a DocBook project](#) on page 560).
6. Click **OK** to dismiss the dialog.

When you click **OK** on the *Set Up DocBook Project* dialog, **Mif2Go** copies a new project configuration file, `_m2docbook.ini`, to your project directory. In addition to the settings you specified in the set-up dialog, this file contains a series of empty configuration sections. *It is up to you to fill these sections with the rest of the settings required to convert your document.* Use a text editor to edit `_m2docbook.ini`; see §4.1 [Working with Mif2Go configuration files](#) on page 91.

17.2.2 Choosing set-up options for a DocBook project

When you choose DocBook as the output type for a new project, the *Set Up* dialog shown in [Figure 17-1](#) opens. [Table 17-1](#) shows the corresponding settings in the configuration file. *You must edit the configuration file to specify additional options.*

See also:

§3.4 [Choosing project set-up options](#) on page 79

§13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425

Figure 17-1 Set Up DocBook Project

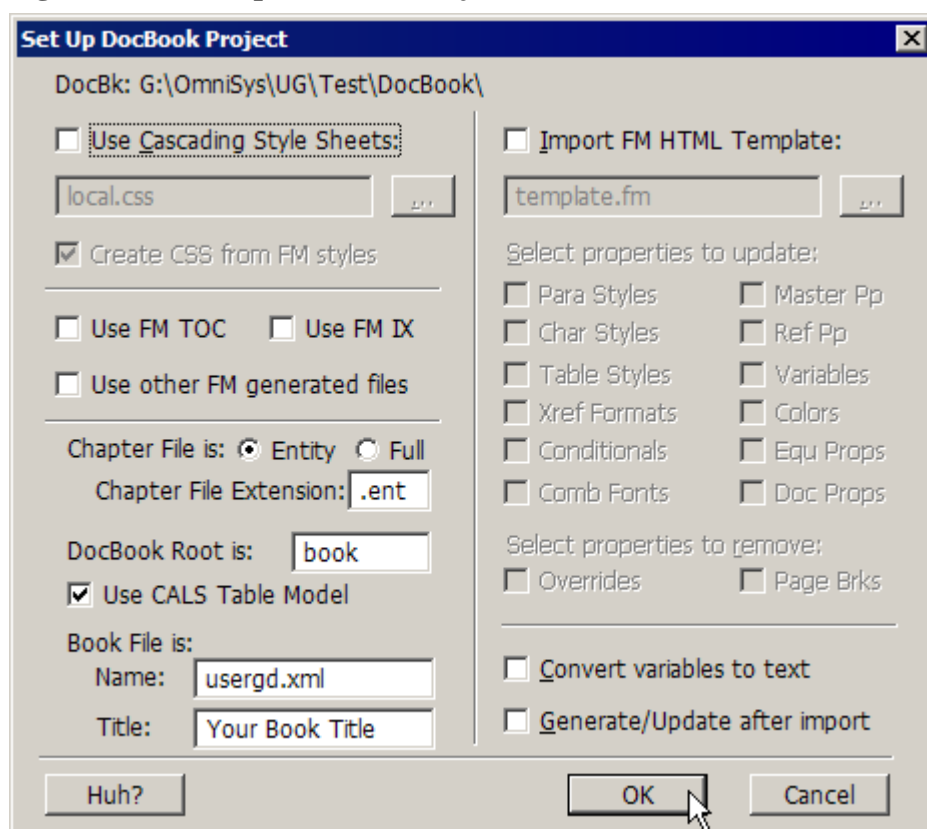


Table 17-1 DocBook set-up options and configuration settings

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Use Cascading Style Sheets	[CSS]	UseCSS=Yes	No	22.4
<i>Path to CSS file</i>	[CSS]	CssFileName= <i>mycss.css</i>	<i>local.css</i>	22.4.3
Create CSS from FM styles	[CSS]	WriteCssStylesheet=0	Never	22.4.3
DocBook Root is:	[DocBookOptions]	DocBookRoot= book article	book	
Use CALS Table Model	[Tables]	UseCALSTableModel=Yes	Yes	17.3.4
Chapter File is:	(None)	<i>See below *</i>	Entity	17.2.3
Chapter File Ext.	[Setup]	XMLSuffix=.ext	.ent	17.2.3
Book File Name	[DocBookOptions]	BookFileName= <i>nm.xml</i>	<i>mydoc.xml</i>	17.2.3
Book File Title	[DocBookOptions]	BookFileTitle= <i>Title</i>	Your Book Title	17.2.3

* Selects the root (book or article) and the chapter-file extension (.ent or .xml). If the root is book, do not check Use Cascading Style Sheets.

17.2.3 Specifying DocBook output options

When you set up a DocBook project, **Mif2Go** includes settings for several output options. You can specify values for these options and a few more in configuration file `_m2docbook.ini`.

In this section:

§17.2.3.1 [Changing the DocBook output file extension](#) on page 561

§17.2.3.2 [Specifying content model and root element](#) on page 561

§17.2.3.3 [Specifying book file options](#) on page 562

17.2.3.1 Changing the DocBook output file extension

To change the file extension for DocBook chapter files:

```
[Setup]
FileSuffix = .ext
```

The default extension is `.ent`.

17.2.3.2 Specifying content model and root element

By default, **Mif2Go** uses the built-in DocBook version 4.5 content model. To specify a different DocBook content model:

```
[DocBookOptions]
; ContentModel = name of content-model .ini, without extension,
; with which to replace the built-in DocBook 4.5 content model.
ContentModel = docbook45
```

If you specify a different content model, you must generate a configuration file for that model from the DTD. See §32 [Working with content models](#) on page 905.

To specify the root element:


```
[DocBookOptions]
; DocBookRoot = element to use at the XMLRoot.
DocBookRoot = book
```

Content models for both <article> and <book> are built in. You can use others, such as <set>, by replacing the content model and specifying the root name here. See §32.2.2 [Generating a content model from a DTD](#) on page 906.

17.2.3.3 Specifying book file options

By default, **Mif2Go** writes a DocBook book file for the FrameMaker book, with the DocBook chapter files included as entity references.

To omit the book file:

```
[DocBookOptions]
; WriteBookFile = Yes (default, write one file for Frame book) or No
; Sets UseDOCTYPE and UseXMLRoot to No for the chapter entity files.
WriteBookFile = No
```

See §17.3.2 [Configuring entity information for DocBook XML](#) on page 563.

To specify the name of the DocBook book file and title of the book:

```
[DocBookOptions]
; BookFileName = name to use for book file, with ext, default is
; Frame book file name with extension .xml
BookFileName = YourBookFileName.xml
; BookFileTitle = text of title for book, default is literally
; Your Book Title
BookFileTitle = Your Book Title
```

17.3 Specifying general options for DocBook

This section lists DocBook-specific default values and recommended options for XHTML and XML configuration settings.

In this section:

- §17.3.1 [Configuring styles for DocBook XML](#) on page 562
- §17.3.2 [Configuring entity information for DocBook XML](#) on page 563
- §17.3.3 [Configuring links for DocBook XML](#) on page 563
- §17.3.4 [Configuring tables for DocBook XML](#) on page 563
- §17.3.6 [Configuring footnotes for DocBook XML](#) on page 564

17.3.1 Configuring styles for DocBook XML

By default, **Mif2Go** suppresses FrameMaker autonumbers for DocBook:

```
[HTMLParaStyles]
* = NoAnum
```

However, if you use FrameMaker numbering properties for things that have nothing to do with numbering, you can override the suppression for selected paragraph formats:

```
[HTMLParaStyles]
ParaFmt = Anum
```

By default, **Mif2Go** converts forced returns (FrameMaker **Shift+Enter**) to spaces for DocBook. To simply close then reopen the paragraph tag (without attributes) instead:

```
[HTMLOptions]
XMLBreakPara = Yes
```

To do so selectively by paragraph format:

```
[HTMLParaStyles]
ParaFmt = XMLBreak
```

See §21.3.8 [Deciding how to treat forced returns](#) on page 651.

17.3.2 Configuring entity information for DocBook XML

Set the following options to values appropriate for DocBook:

```
[HTMLOptions]
; UseDOCTYPE = Yes (default) or No (when writing DocBook entity files)
UseDOCTYPE = No
; UseXMLRoot = Yes (default) or No (when writing DocBook entity files)
UseXMLRoot = No
; XHLangAttr = xml:lang (default, set as needed)
XHLangAttr = xml:lang
```

When `[DocBookOptions]WriteBookFile=Yes`, **Mif2Go** sets `UseDOCTYPE` and `UseXMLRoot` to `No` for chapter entity files; see §17.2.3 [Specifying DocBook output options](#) on page 561.

17.3.3 Configuring links for DocBook XML

DocBook requires linking mark-up that is slightly different from mark-up for generic XML. The following settings require other than the default value:

```
[HTMLOptions]
; HrefAttribute = name to use for link source attr, default href; use
; linkend for DocBook
HrefAttribute = linkend
; UseHash = Yes (default, start local hrefs with #)
; or No; the # is not valid in DocBook
UseHash = No
; UseUlink = No (default, use ATagName for URLs) or Yes (use
; ulink for URLs, and url as the HrefAttribute within them)
UseUlink = Yes
; RemoveXrefHotspots = No (default) or Yes (remove hotspot text for
; xrefs and hyperlinks to Frame files, retain it for external URLs)
RemoveXrefHotspots = Yes
; UseListedXrefFilesOnly = No (default) or Yes (consider any xref
; target files not listed in [XrefFiles] to refer to the current
; file.) This suppresses filenames for DocBook where files are in the
; same DocBook book; files not in the book must be listed in
; [XrefFiles].
UseListedXrefFilesOnly = Yes
```

See also:

§14.6 [Configuring links for generic XML](#) on page 467

17.3.4 Configuring tables for DocBook XML

DocBook supports both the CALS table model and the HTML table model:

```
[Tables]
; UseCALSModel = No (HTML default) or Yes (XML default)
UseCALSModel = Yes
; UseInformaltableTag = No (default) or Yes (use when there is no
; table caption)
UseInformaltableTag = Yes
; InternalTableCaption = Yes (default) or No (put outside table)
InternalTableCaption = No
```

```
; TableCaptionTag = tag for internal table captions, default "caption"
TableCaptionTag = caption
```

17.3.5 Retaining empty paragraph tags in DocBook table cells

By default, for DocBook output **Mif2Go** omits paragraph tags from otherwise empty non-preformatted paragraphs in table cells. However, you can choose to keep the tags:

```
[Tables]
; RemoveEmptyTableParagraphs = No (default)
; or Yes (DITA/DocBook default)
RemoveEmptyTableParagraphs = No
```

When `RemoveEmptyTableParagraphs=No`, paragraph tags (such as `<para></para>`) for empty paragraphs are retained in table cells in DocBook XML.

When `RemoveEmptyTableParagraphs=Yes`, paragraph tags for empty paragraphs in table cells are omitted (except for preformatted text, where tags are always preserved). A table cell that is blank in FrameMaker (contains only empty paragraphs) would become just `<entry></entry>` in DocBook XML output.

Note: This setting is independent of the setting for removing empty paragraphs in text; see §21.3.10 [Eliminating empty paragraphs in text](#) on page 652.

See also:

§24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744

17.3.6 Configuring footnotes for DocBook XML

Footnotes in DocBook require other than the default settings for some features:

```
[HTMLOptions]
; Footnotes = Jump (HTML default, at end), Embed (between []),
; Inline (XML default), or None
Footnotes = Inline
; FootInlineParaTag = tag for beginning and ending inline footnote
; paras
FootInlineParaTag = para
; FootInlineIDPrefix = start of ID attr for inline footnotes; rest
; is sequential number starting with 1 at start of file.
FootInlineIDPrefix = foot
; UseFootXrefTag = No (HTML default) or Yes (XML default)
UseFootXrefTag = Yes
; FootInlineRefTag = tag for xrefs to inline footnotes, uses linkend
; for href attribute, for DocBook
FootInlineXrefTag = footnoteref
```

See also:

§21.11 [Converting footnotes to HTML or XML](#) on page 671

17.4 Configuring DocBook elements

In this section:

§17.4.1 [Treating FrameMaker format names as element names](#) on page 565

§17.4.2 [Mapping paragraph formats to DocBook elements](#) on page 565

§17.4.3 [Mapping character formats to DocBook elements](#) on page 568

§17.4.4 [Assigning ID attributes to DocBook block elements](#) on page 569

§17.4.5 [Assigning attributes other than ID to DocBook elements](#) on page 571

17.4.1 Treating FrameMaker format names as element names

If some of your FrameMaker formats are already named for DocBook elements, you can lessen the chore of mapping formats to elements by directing **Mif2Go** to use the format name as the DocBook element name wherever possible (that is, when the content model includes an element of that name). This works only if the named element is of an appropriate type: block allowing text for a paragraph format, or inline allowing text for a character format.

However, leaving any paragraph format unmapped is risky; some formats might match the names of DocBook elements that do not do what you want.

To map FrameMaker format names to DocBook elements of the same name, where possible:

```
[DocBookOptions]
; UseFormatAsTag = No (default, if tag unmapped use default elem),
; or Yes (if unmapped, use Frame format name if valid in content
; model).
UseFormatAsTag = Yes
```

When `UseFormatAsTag=Yes`, any FrameMaker format with a name that is the same as a DocBook element name in the current content model is mapped to that element.

Unmapped format names that do not correspond to such element names are mapped to the default element; see:

§17.4.2.2 [Specifying a default element for unmapped paragraph formats](#) on page 566

§17.4.3.2 [Specifying a default element for unmapped character formats](#) on page 569.

17.4.2 Mapping paragraph formats to DocBook elements

When you map paragraph formats to DocBook elements, you must ensure that the element mapped to is allowed to contain text.

In this section:

§17.4.2.1 [Assigning DocBook elements to paragraph formats](#) on page 565

§17.4.2.2 [Specifying a default element for unmapped paragraph formats](#) on page 566

§17.4.2.3 [Omitting invalid tags for default DocBook block elements](#) on page 566

§17.4.2.4 [Overriding element mapping for paragraph formats](#) on page 567

§17.4.2.5 [Providing aliases for paragraph formats](#) on page 567

17.4.2.1 Assigning DocBook elements to paragraph formats

To map paragraph formats in your document to DocBook elements, assign the element name to the format name:

```
[DocBookParaTags]
; Frame paragraph format (wildcards OK) = DocBook element, can be
; overridden by a DocBookTag marker; or Frame format = No.
ParaFmtName = elementname
```

Default element The default element for a FrameMaker paragraph format that is not mapped in `[DocBookParaTags]` is one of the following:

- If `UseFormatAsTag=Yes` and the name of the format matches the name of a DocBook element, the format is mapped to that element.
- If `UseFormatAsTag=No` or the format name does not match an element name, the format is mapped to the element designated by `DefParaElem`; see §17.4.2.2 [Specifying a default element for unmapped paragraph formats](#) on page 566.

Specify ancestry for list formats For list formats, if mapping the format to an element is not sufficient to identify the list type, you must also specify the parent of the element; see §17.5.2 [Designating DocBook ancestor elements](#) on page 573. Definition lists can be derived from paragraph pairs, possibly with run-in headings for the term.

Omit element mapping To specify that a particular FrameMaker paragraph format should *not* be mapped to any element:

```
[DocBookParaTags]
ParaFmtName = No
```

The value `No` means that the tags for the format should be omitted, leaving the text inside the enclosing element. Use this mapping for code examples (which can run on for pages), to avoid having each line of code mapped to a separate `<codeblock>` element. For example:

```
[DocBookParaTags]
PgmCode* = No

[DocBookParents]
PgmCode* = codeblock
```

Specifying ancestry guarantees that **Mif2Go** will retain the original line breaks, instead of normalizing them as for HTML or XML.

See §17.5.2 [Designating DocBook ancestor elements](#) on page 573.

17.4.2.2 Specifying a default element for unmapped paragraph formats

To specify a default element to use for unmapped paragraph formats:

```
[DocBookOptions]
; DefParaElem = element to use for Frame para formats that are neither
;   named for DocBook elements nor mapped in [DocBookParaTags].
DefParaElem = para
```

If your configuration file does not include a value for `DefParaElem`, **Mif2Go** uses one of the following as the element for an unmapped format: if `UseFormatAsTag=Yes` and the FrameMaker format name (adjusted as for CSS class names) matches the name of a valid element in the current content model, the format is mapped to that element; otherwise, the format is mapped to `para`, the default value of `DefParaElem`. See §17.4.1 [Treating FrameMaker format names as element names](#) on page 565.

17.4.2.3 Omitting invalid tags for default DocBook block elements

Some DocBook block elements allow only `#PCDATA`, not paragraph tags. When a “normal” paragraph must be placed inside one of these blocks, the paragraph tag should be omitted.

If some paragraph formats in your FrameMaker document are left unmapped, or are explicitly mapped to the default block element (usually `<para>`), the presence of such paragraphs in contexts where the default block element would not be valid could trigger unwanted interpolation of an arbitrary parent element. For enclosing block elements that allow mixed content, you can avoid this problem by directing **Mif2Go** to omit the default paragraph tags.

To omit invalid default paragraph tags where mixed content is allowed:

```
[DocBookOptions]
; DropInvalidParaTag = No (default) or Yes (if the para tag is the
;   default DefParaElem <para> and is invalid, but #PCDATA is valid,
;   drop the tag)
DropInvalidParaTag = Yes
```

See also:

§17.5.3 [Fixing up interpolated ancestries](#) on page 574

17.4.2.4 Overriding element mapping for paragraph formats

To override the element-name mapping for a given paragraph, insert a **DocBookTag** marker in the paragraph, with content the desired element name.

If mapping (or overriding mapping) does not suffice, and you do not need to specify a required ancestry for the element, use the following instead:

- [HTMLParaStyles] CodeBefore and CodeAfter properties for the format
- [ParaStyleCodeBefore] and [ParaStyleCodeAfter] sections to specify the element tags to surround the text.

See §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

Another alternative would be to bracket the text with **Config** markers, with content such as [ParaStyleCodeBefore]=<element> and [ParaStyleCodeAfter]=</element>; see §33.2.2 [Overriding settings with configuration markers](#) on page 921.

Note: Mappings provided via [ParaStyleCode*] settings or markers do not participate in any ancestry you specify for the element in question; see §17.5 [Nesting DocBook block elements](#) on page 573.

17.4.2.5 Providing aliases for paragraph formats

If you are generating DocBook from an unstructured FrameMaker document, your document might use the same format for different purposes, each purpose requiring that format to be mapped to a different DocBook element, or to be nested in a different hierarchy, or both; or you might have several formats that map to the same DocBook element. Because DocBook uses semantic tags, whereas FrameMaker uses presentational tags, in some cases you need alternate names for paragraph formats to clarify semantic use cases.

To specify an alternate name, or alias, for a paragraph format:

```
[DocBookAlias]
; Frame paragraph format = Frame format name to use in place of that
; paragraph format for DocBook purposes
ParaFmtName = AlternateName
```

An alias works in any [DocBook*] configuration section that uses format names. The alias can be the name of another paragraph format in your document, provided the two formats map to exactly the same element with all the same DocBook settings; or, the alias can be a name you invent.

For additional aliases for the same format, insert a **DocBookAlias** marker in each instance of the format that requires a different alias, with content the name of another alias. You can also use a **DocBookAlias** marker to override an alias assigned in section [DocBookAlias].

You can use as many different aliases for the same paragraph format as your document requires. If you are creating new alias names, be careful not to duplicate the name of a format that is already in the FrameMaker paragraph catalog.

You might need to create aliases in the following situations:

[One-to-many mappings](#) of the same format to different DocBook elements

[Many-to-one mappings](#) of two or more formats to the same DocBook element.

One-to-many mappings

Suppose your FrameMaker document includes a paragraph format named *Body2*, used in the following situations:

- most often as a continuation of a *Numbered1* or *Numbered* paragraph
- less often as a continuation of a *Bulleted* paragraph
- occasionally as a quotation, not part of any list.

This means that in different places in your document *Body2* would have to be mapped to different elements, or participate in different DocBook hierarchies.

To resolve this conflict, you would assign aliases to the alternate uses of *Body2*. You could keep the original format name for the most frequent use; however, the name *Body2* does not convey anything about the differing semantics. Therefore you might want to use aliases for every use; for example, *Body2OList*, *Body2IList*, and *Body2Quote*.

To create an alias for the most prevalent use of *Body2*:

```
[DocBookAlias]
Body2 = Body2OList
```

For the other two uses of *Body2*, you would have to insert a **DocBookAlias** marker in each instance, with content one of the other aliases: *Body2IList* or *Body2Quote*. Then you could specify the following in your project configuration file:

```
[DocBookTags]
Body2?list = para
Body2Quote = blockquote
```

Many-to-one mappings

Suppose your FrameMaker document includes three different paragraph formats for quotations:

Quote in body text
FtnQ in footnotes
CellQ in table cells.

All three map to DocBook element `<blockquote>`. You can make this semantic equivalence explicit in section `[DocBookAlias]`, and use the collective alias in other configuration sections:

```
[DocBookTags]
Quote = blockquote

[DocBookAlias]
FtnQ = Quote
CellQ = Quote
```

17.4.3 Mapping character formats to DocBook elements

When you map character formats to DocBook elements, make sure that the element mapped to is allowed to contain text.

In this section:

- §17.4.3.1 [Assigning DocBook elements to character formats](#) on page 568
- §17.4.3.2 [Specifying a default element for unmapped character formats](#) on page 569
- §17.4.3.3 [Overriding element mapping for character formats](#) on page 569

17.4.3.1 Assigning DocBook elements to character formats

To map character formats in your document to DocBook elements, assign an element name to each character format name:

```
[DocBookCharTags]
; Frame character format (wildcards OK) = DocBook element, cannot be
```



```
; overridden by a DocBookTag marker; or Frame format = No.
CharFmtName = elementname
```

To specify that a particular FrameMaker character format should *not* be mapped to an element:

```
[DocBookCharTags]
CharFmtName = No
```

The value `No` means that tags for the format should be omitted, leaving the text inside the enclosing element. For example, map the character formats you use for links and cross references to `No`. **Mif2Go** automatically generates `<xref>` tags from the cross references in FrameMaker, based on the format, but you do not need to map the format itself to any element.

The default element for a FrameMaker character format that is not mapped in `[DocBookCharTags]` is the element designated by `DefCharElem`; see §17.4.3.2 [Specifying a default element for unmapped character formats](#) on page 569. It is best to map each character format to the most specific element possible, which is not often the default element.

17.4.3.2 Specifying a default element for unmapped character formats

To specify a default element to use for unmapped character formats:

```
[DocBookOptions]
; DefCharElem = element for Frame char formats that are neither
; named for DocBook elements nor mapped in [DocBookCharTags]
DefCharElem = phrase
```

If your configuration file does not include a value for `DefCharElem`, **Mif2Go** uses one of the following as the element for an unmapped format: if `UseFormatAsTag=Yes` and the FrameMaker format name (adjusted as for CSS class names) matches the name of a valid element in the current content model, the format is mapped to that element; otherwise, the format is mapped to `phrase`, the default value of `DefCharElem`. See §17.4.1 [Treating FrameMaker format names as element names](#) on page 565.

17.4.3.3 Overriding element mapping for character formats

If mapping a FrameMaker character format does not suffice for an inline element, you can use **DocBookStartElem** and **DocBookEndElem** markers placed at the start and end, respectively, of the character span to be delimited as an element. The content of each marker is the tag name for the inline element; **Mif2Go** provides the `< >` and `</ >`. You cannot use a **DocBookTag** marker to override the element-name mapping for an inline element.

17.4.4 Assigning ID attributes to DocBook block elements

Every block element in DocBook that is the target of a cross reference or hypertext link must have an ID attribute. You can have **Mif2Go** automatically assign an ID to each block element derived from a FrameMaker paragraph (or to an interpolated parent of such a block element). You can also use markers to assign IDs to specific block elements, or to override a **Mif2Go**-assigned ID.

In this section:

§17.4.4.1 [Understanding how Mif2Go creates ID attribute values](#) on page 570

§17.4.4.2 [Providing IDs for block elements](#) on page 570

§17.4.4.3 [Providing IDs for interpolated parents of block elements](#) on page 570

§17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571

17.4.4.1 Understanding how Mif2Go creates ID attribute values

When **Mif2Go** assigns an ID to a block element in DocBook, the value of the ID attribute is a combination of the **Mif2Go** FileID of the FrameMaker file being processed and the FrameMaker ObjectID of the paragraph from which the element was generated; see §5.1.10 [Preserving Word-generated cross-reference markers](#) on page 114.

If a particular block element requires an ID value, **Mif2Go** looks for a **DocBookElemID** marker in the paragraph from which the element was generated. If that paragraph does not contain a **DocBookElemID** marker, **Mif2Go** uses the content of the first **newlink** marker in the paragraph as the ID for the element. If there is no **newlink** marker, and you have requested automatic ID assignment for that element, **Mif2Go** assigns an ID.

You can override an automatically assigned ID for a particular block element by inserting either a **newlink** marker or a **DocBookElemID** marker in the paragraph from which the element is generated; see §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.

17.4.4.2 Providing IDs for block elements

To direct **Mif2Go** to automatically include an ID attribute in each instance of a block element mapped from a particular FrameMaker paragraph format:

```
[DocBookParaIDs]
; Frame para format = No (default, no ID set automatically) or Yes
ParaFmt = Yes
```

When *ParaFmt*=Yes, **Mif2Go** includes an ID attribute in the element generated from each paragraph in that format. The default is not to include an ID attribute.

When *ParaFmt*=No (or when no [DocBookParentIDs] setting is present for *ParaFmt*), **Mif2Go** does not include an ID attribute in the elements generated. For an individual element that is the target of a link or cross reference, unless a **newlink** marker is already present in the FrameMaker paragraph from which the element was generated, you must insert a **DocBookElemID** marker to provide an ID for the element; see §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.

Note: If you list the same paragraph format in [DocBookParentIDs], **Mif2Go** changes the value in [DocBookParaIDs] to No; see §17.4.4.3 [Providing IDs for interpolated parents of block elements](#) on page 570.

You can override the value of an automatically assigned ID attribute with a **newlink** marker or a **DocBookElemID** marker inserted in the paragraph in FrameMaker; see §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.

17.4.4.3 Providing IDs for interpolated parents of block elements

To direct **Mif2Go** to include an ID attribute in each instance of an interpolated parent of an element mapped from a particular FrameMaker paragraph format:

```
[DocBookParentIDs]
; Frame para format = single parent element for which the Frame ID
; of the para should be used.
ParaFmt = ParentElement
```

When you assign a parent element in [DocBookParentIDs], **Mif2Go** includes in the specified interpolated parent of each element mapped from *ParaFmt* the ID attribute assigned (or that would have been assigned) to that element in [DocBookParaIDs]. The

ID attribute is added only if the parent is interpolated by **Mif2Go** as a required ancestor of the current element. Because IDs must be unique, an automatically assigned parent ID disables any **Yes** setting for the same format in [DocBookParaIDs]; see §17.4.4.2 [Providing IDs for block elements](#) on page 570. And it also eliminates any ID assigned to the child via **DocBookElemID** marker; see §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.

You can override the value of an automatically assigned parent ID attribute with a **DocBookParentID** marker inserted in the child paragraph in FrameMaker; see §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.

17.4.4.4 Specifying an ID for an individual block element or parent

To specify an ID for a single instance of a block element, place a **DocBookElemID** marker in the FrameMaker paragraph. The content of the marker is the value of the `id` attribute. You can also override any **Mif2Go**-assigned ID with a **DocBookElemID** marker.

To specify an ID for the interpolated parent of the block element derived from a particular paragraph, place a **DocBookParentID** marker in the paragraph, with content as follows:

```
parentname=parentid
```

Do not include spaces around the equals sign.

See also:

§17.4.4.2 [Providing IDs for block elements](#) on page 570

17.4.5 Assigning attributes other than ID to DocBook elements

You can include non-ID attributes in a DocBook block or inline element by assigning `attribute="value"` pairs to the FrameMaker format mapped to the element. The attributes you assign with configuration settings apply to all instances of the element in question. Only those attributes assigned to elements mapped from paragraph formats can be overridden with markers; attributes of elements mapped from character formats cannot be overridden with markers.

In this section:

§17.4.5.1 [Specifying attribute values for a block element or ancestor](#) on page 571

§17.4.5.2 [Specifying attribute values for an inline element](#) on page 572

17.4.5.1 Specifying attribute values for a block element or ancestor

You can do any of the following for block elements:

[Assign block element attributes](#)

[Override block element attributes](#)

[Assign interpolated parent attributes](#)

[Override interpolated parent attributes](#)

When you want to override default or assigned attributes, keep in mind:

[Where to use DocBook Attribute markers](#)

Assign block element attributes

To apply attributes (other than `id`) to a block element (other than `<xref>`), assign `attribute="value"` pairs, separated by spaces, to the paragraph format(s) mapped to the element:

```
[DocBookParaAttributes]
; Frame para format (wildcards OK) = attributes
ParaFmt = attribute1="value1" attribute2="value2" ...
```

You can use **Mif2Go** macros for any part of the assignment, or even for the entire assignment. For example:

```
[DocBookParaAttributes]
ParaFmt = <$WriteAttrMacro>
```

*Override block
element attributes*

To override a setting in [DocBookParaAttributes] or to override default attributes for a particular instance of a block element, place a **DocBookAttribute** marker in a paragraph mapped to the element, with content as follows:

```
elementname: attribute1="value1" attribute2="value2" ...
```

For example:

```
step: performance="optional"
```

The name of the element must be followed by a colon. Separate *attribute="value"* pairs with a space. Each value must be enclosed in double quotes. You can use **Mif2Go** macros for everything after the colon.

*Assign
interpolated
parent attributes*

To assign attributes to an interpolated parent of a block element:

```
[DocBookParentAttributes]
; Frame para format (wildcards OK) = parentname: attributes
ParaFmt = parentname: attribute1="value1" attribute2="value2" ...
```

You can use **Mif2Go** macros for the assignment.

*Override
interpolated
parent attributes*

To override a setting in [DocBookParentAttributes] or to override default attributes for an interpolated parent of a block element, place a **DocBookAttribute** marker in a paragraph mapped to the element, with content as follows:

```
parentname: attribute1="value1" attribute2="value2" ...
```

To apply attributes to more than one interpolated parent, use a separate marker for each parent.

*Where to use
DocBook
Attribute markers*

Use **DocBookAttribute** markers only to supply attribute values other than the DTD default values for an element, or to override attribute values specified in a configuration file. Do *not* use **DocBookAttribute** markers for either of the following:

- The *id* attribute of the current element; use a **DocBookElemID** marker instead. See §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.
- The *id* attribute of an interpolated parent of the current element; use a **DocBookParentID** marker instead. See §17.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 571.

A **DocBookAttribute** marker overrides settings in [DocBookParaAttributes] and [DocBookParentAttributes], but does not override settings in [DocBookCharAttributes] (see §17.4.5.2 [Specifying attribute values for an inline element](#) on page 572).

17.4.5.2 Specifying attribute values for an inline element

To apply attributes (other than *id*) to an inline element, assign *attribute="value"* pairs, separated by spaces, to the character format(s) mapped to the element:

```
[DocBookCharAttributes]
; Frame char format (wildcards OK) = attributes
CharFmt = attribute1="value1" attribute2="value2" ...
```

You cannot use markers to override settings in [DocBookCharAttributes].

17.5 Nesting DocBook block elements

Nesting block elements is the most challenging aspect of treating unstructured FrameMaker as though it were structured.

In this section:

§17.5.1 [Understanding how Mif2Go determines element nesting](#) on page 573

§17.5.2 [Designating DocBook ancestor elements](#) on page 573

§17.5.3 [Fixing up interpolated ancestries](#) on page 574

§17.5.4 [Deciding when to fully specify ancestry](#) on page 575

§17.5.5 [Specifying alternate ancestries for the same element](#) on page 575

§17.5.6 [Specifying first-child status for nested elements](#) on page 576

§17.5.7 [Specifying full ancestry for nested sections](#) on page 576

§17.5.8 [Closing DocBook ancestor elements](#) on page 577

§17.5.9 [Opening DocBook ancestor elements](#) on page 578

§17.5.10 [Configuring multi-paragraph list items](#) on page 578

§17.5.11 [Specifying DocBook element levels](#) on page 579

See also:

§17.6 [Designating ancestors for table elements](#) on page 580

§17.7 [Specifying options for figure elements](#) on page 581

17.5.1 Understanding how Mif2Go determines element nesting

For each element, **Mif2Go** considers whether that element can go inside the current parent element. If not, **Mif2Go** uses heuristic methods based on the possible parents, level limitations, and current context.

For example, suppose your document uses a sequential structure for steps in a procedure: paragraph format *Step1* for the first step, followed by several *StepNext* paragraphs. To convert this structure to a hierarchical DocBook structure, with paragraphs in both formats becoming `<step>` children of a `<procedure>` element, you would specify just one setting (see §17.4.2 [Mapping paragraph formats to DocBook elements](#) on page 565):

```
[DocBookParaTags]
Step* = step
```

As soon as **Mif2Go** encounters a paragraph format that is not valid in `<procedure>`, the parent tag is closed.

For problem cases, you can use a **DocBookLevel** marker to explicitly set the level for an element; see §17.5.11 [Specifying DocBook element levels](#) on page 579. However, for nested lists, use a different approach; see §17.5.5 [Specifying alternate ancestries for the same element](#) on page 575.

Leaving *any* paragraph or character format unmapped to a parent is risky; **Mif2Go** might interpolate the name of a DocBook element that does not do what you want.

17.5.2 Designating DocBook ancestor elements

For block elements such as `<listitem>` that can have more than one possible ancestry, map any paragraph formats to the intended (required) parent element, and if necessary, grandparent element, even great-grandparent element. List ancestors in hierarchical order. For example:

```
[DocBookParents]
; Frame para format (wildcards OK) = required parents
Heading* = section
Numbered1 = orderedlist listitem
Numbered = orderedlist listitem
Bulleted = itemizedlist listitem
TableTitle = table
Figure Title = mediaobject
Example = example
```

These settings specify, for example, that a *Numbered1* paragraph (which is mapped to `<para>` in `[DocBookParaTags]`) has these ancestors:

```
<orderedlist>...<listitem>...</listitem>...</orderedlist>
```

Therefore, each *Numbered1* paragraph starts a new `<orderedlist>` if and only if an `<orderedlist>` is not already open; and starts a new `<listitem>` if and only if an `<listitem>` under the `<orderedlist>` is not already open. To force a new `<orderedlist><listitem>` for *Numbered1* paragraphs, you must also give the *Numbered1* paragraph format first-child status under both parent and grandparent elements; see §17.5.6 [Specifying first-child status for nested elements](#) on page 576.

Note: For list items that can include more than one paragraph, map the paragraph format to `<para>`, then designate its including list element as a parent.

Use this mapping for formats such as lists, in which `<listitem>` elements are needed under `<itemizedlist>` or `<orderedlist>` in addition to the `<para>` elements mapped in `[DocBookParaTags]`.

List ancestors in hierarchical order

If a parent element has more than one possible parent, and only one of those parents can be a grandparent of the paragraph format in question, list both the grandparent and parent, in hierarchical order.

Override individual ancestries

To override the `[DocBookParents]` assignment for a given instance of a paragraph format, place a **DocBookParent** marker in the paragraph. Make the content of the marker the name(s) of the ancestor element(s), in hierarchical order. A **DocBookParent** marker specifies the required ancestry for the current block element, overriding whatever is specified in `[DocBookParents]`.

17.5.3 Fixing up interpolated ancestries

Creating DocBook structure from FrameMaker formats necessarily involves some trial and error. When you see unexpected interpolation of inappropriate parent elements in your output, it is usually because you have not specified parents for a particular format-to-element mapping. For example, suppose you map paragraph format *Ref* to `<para>`, and use a *Ref* paragraph at the top level of each reference section, where `<para>` is not valid. On encountering a *Ref* paragraph in this situation, with no parents specified for the *Ref* format, **Mif2Go** would go through the list of valid parents for `<para>` in a reference section, and interpolate the first set that works.

The remedy is to figure out what would be a more appropriate lineage for the element in question. You could specify that lineage for the format in `[DocBookParents]` if it applies generally, or insert a **DocBookParent** marker in the paragraph for an isolated instance. In this example, the following mapping would produce better results:

```
[DocBookParents]
Ref = refentry refsect1
```

The **Mif2Go** search algorithm finds the shortest path, but that is not always the only shortest path, or the best path.

See also:

§17.4.2.3 [Omitting invalid tags for default DocBook block elements](#) on page 566

17.5.4 Deciding when to fully specify ancestry

You do not need to map paragraphs in [DocBookParents] for elements that can have only one possible ancestry, or for cases where **Mif2Go** can determine heuristically which of the possible ancestors fits the context best. Specify ancestry in [DocBookParents] when more than one lineage is possible in the context of use. This is especially important if your document includes nested `section` elements; see §17.5.7 [Specifying full ancestry for nested sections](#) on page 576.

Include as many ancestors as necessary to fully specify ancestry for the element to which a paragraph format is mapped in [DocBookParaTags]. If your document includes actual instances of different ancestries for the same element, use sets of ancestors to specify the alternatives; see §17.5.5 [Specifying alternate ancestries for the same element](#) on page 575. In some cases you might have to include all ancestors up to the topic level, and you might have to determine this necessity by trial and error; that is, list them all whenever *not* including all ancestors causes unwanted nesting.

When **Mif2Go** encounters a set of ancestors specified either in [DocBookParents] or in a **DocBookParent** marker, **Mif2Go** tries to nest the ancestor hierarchy in the current element. If the entire hierarchy is valid in that position, that is where it stays. This means that if your FrameMaker document uses paragraph format *Body* (for example) for all text that is not nested in a list, and you map *Body* to DocBook element `<para>`, you must also specify non-list parents for *Body*, because `<para>` can nest in `<listitem>`; in fact, in almost any block element. Unless you can make sure every block element that could precede a *Body* paragraph gets closed (see §17.5.8 [Closing DocBook ancestor elements](#) on page 577), the *Body* `<para>` is likely to be nested in the preceding element.

17.5.5 Specifying alternate ancestries for the same element

If your document uses the same paragraph format in several lineages, you can create a set of alternate ancestors for **Mif2Go** to choose from, depending on the context. The following predefined element sets are included in your project configuration file when you first set up a DocBook project. You can alter or delete these sets, and you can define additional sets.

To define sets of elements to be considered as alternate ancestors:

```
[DocBookElementSets]
; $setname = DocBook elements in the set.
; These element sets are predefined in the starting .ini for DocBook:
$top = chapter appendix preface article
$sections = sect1 sect2 sect3 sect4 sect5 section simplesect
$text = sect1 sect2 sect3 sect4 sect5 section simplesect chapter
appendix preface article entry
$list = itemizedlist orderedlist
```

Each set name must start with a dollar sign (\$). You must define each set as a collection of elements; you may *not* define one element set in terms of other element sets. The list of elements in the set must be all on the same line, even if it does not appear that way here.

Note: Element set `$list` does not include element `simplelist`, because `simplelist` is more restricted as to content than the other list types.

You can use an element set name in place of an element name in [DocBookParents], in [DocBookFirst], or in the corresponding **DocBookParent** and **DocBookFirst** markers. For example:

```
[DocBookParents]
Body = $text
Body2 = $text $list listitem
```

Any element in the set is acceptable at the point where it appears in the hierarchical sequence. There is no equivalent marker.

17.5.6 Specifying first-child status for nested elements

To specify parent elements in which the paragraph format mapped to a given block element must appear as the first child:

```
[DocBookFirst]
; Frame para format = parents under which the current block element
; (or one of its parents) must be the first child.
Numbered1 = orderedlist listitem
Numbered = listitem
Bulleted = listitem
```

If the parent element you assign to a paragraph format has more than one possible parent, and the paragraph format in question needs to be first only for one of its possible grandparents, list both the grandparent and parent, separated by spaces. You can list as many ancestors as necessary to fully specify first-child status for the paragraph format. List the ancestors in hierarchical order. The list must match the ancestor list in [DocBookParents]; see §17.5.2 [Designating DocBook ancestor elements](#) on page 573.

Use these settings mainly for lists, to ensure that a paragraph format starts a new list item or a new list. For example, these settings specify the following for the list paragraph formats mapped to <para> in [DocBookParaTags]:

- A *Numbered1* <para> element must be the first child of its parent <listitem> element, which <listitem> element must be the first child of its <orderedlist> parent; this setting forces first-child status for the entire lineage of the elements listed, not just the last.
- A *Numbered* <para> element or a *Bulleted* <para> element must be the first child of its parent <listitem> element.

To override the [DocBookFirst] assignment for a given instance of a paragraph, place a **DocBookFirst** marker in the paragraph. Make the content of the marker the name(s) of the desired ancestor element(s), in hierarchical order. A **DocBookFirst** marker specifies that the current block element must be the first child of its listed ancestor elements, overriding whatever is specified in [DocBookFirst].

17.5.7 Specifying full ancestry for nested sections

When you have nested DocBook sections you must specify parentage starting with \$top for every section title. For example:

```
[DocBookParents]
Heading1 = $top section
Heading2 = $top section section
Heading3 = $top section section section
Heading4 = $top section section section section
```

Otherwise, the higher levels would also match the rule for the lower levels; so, for example, the following settings:

```
[DocBookParents]
Heading1 = section
Heading2 = section section
```

would allow another *Heading1* section to follow a *Heading2* section without closing the lower-level *Heading2* section. The starting `$top` prevents this.

In addition, you would need to specify:

```
[DocBookFirst]
Heading* = section
```

so that each heading starts a new section when it occurs at the same level as the preceding section. Otherwise a second *Heading2* section would be valid inside the first *Heading2* section, and would not close that section and start a new section of its own at the same level.

See also:

[§17.5.5 Specifying alternate ancestries for the same element](#) on page 575

[§17.5.6 Specifying first-child status for nested elements](#) on page 576

17.5.8 Closing DocBook ancestor elements

To get a block element under the correct parent, you might have to specify that an ancestor element (and all its descendants) must end when the current block element ends; or that the prior block must end before the current block element begins.

In this section:

[§17.5.8.1 Ending ancestor elements before the current block](#) on page 577

[§17.5.8.2 Ending ancestor elements after the current block](#) on page 577

17.5.8.1 Ending ancestor elements before the current block

In some cases, it is not clear whether a paragraph is supposed to be a child of the preceding element (or nest of elements). For example, by default a `<para>` element following a list item becomes part of the `<listitem>`, and that is not necessarily what you want.

To close an element (or a hierarchy of elements) before starting the current block (for example):

```
[DocBookCloseBefore]
; Frame para format = elements to be closed, with any other elements
;   nested under them, before the current block element starts.
Recap = listitem
Body = itemizedlist orderedlist
```

Use this setting to force closure of elements that were opened based on settings in `[DocBookParents]`; see [§17.5.2 Designating DocBook ancestor elements](#) on page 573. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DocBookCloseBefore** marker in the paragraph for the current block element instead, with content the name(s) of the element(s) to close. You can also use a **DocBookCloseBefore** marker to override a `[DocBookCloseBefore]` setting when you want to close a higher (or lower) ancestor than the setting specifies.

17.5.8.2 Ending ancestor elements after the current block

In some cases, it is not clear whether the end of a block element should also end the enclosing parent element. To close a parent element at the end of the current block element (for example):

```
[DocBookCloseAfter]
; Frame para format = parent to be closed, with any other elements
; nested under it, at the end of the current block element.
FigAnchor = figure
```

Use this setting to force closure of elements that were opened based on settings in [DocBookParents]; see §17.5.2 [Designating DocBook ancestor elements](#) on page 573. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DocBookCloseAfter** marker in the paragraph for the current block element instead, with content the name(s) of the ancestor element(s) to close. You can also use a **DocBookCloseAfter** marker to override a [DocBookCloseAfter] setting when you want to close a higher (or lower) ancestor than the setting specifies.

17.5.9 Opening DocBook ancestor elements

To get a block element in the correct position in a hierarchy, you might have to force the opening of interpolated ancestor elements first; or, in some cases, specify elements that must be opened after the current element ends.

In this section:

§17.5.9.1 [Starting ancestor elements before the current block](#) on page 578

§17.5.9.2 [Starting a new hierarchy after the current block](#) on page 578

17.5.9.1 Starting ancestor elements before the current block

To open interpolated ancestor elements before starting the current block:

```
[DocBookOpenBefore]
; Frame para format = elements to be opened, with any other elements
; nested under them, before the current block element starts.
somefmt=someancestor
```

Use this setting to force opening of elements when [DocBookParents] does not suffice.

For individual cases, you can insert a **DocBookOpenBefore** marker in the paragraph for the current block element instead, with content the name(s) of the element(s) to open. You can also use a **DocBookOpenBefore** marker to override a [DocBookOpenBefore] setting when you want to open a higher (or lower) ancestor than the setting specifies.

17.5.9.2 Starting a new hierarchy after the current block

To open a new element or hierarchy of elements after the current block ends:

```
[DocBookOpenAfter]
; Frame para format = elements to be opened, with any other elements
; nested under them, before the current block element starts.
somefmt=someancestor
```

Use this setting to force opening of elements when [DocBookParents] does not suffice.

For individual cases, you can insert a **DocBookOpenAfter** marker in the paragraph for the current block element instead, with content the name(s) of the element(s) to open. You can also use a **DocBookOpenAfter** marker to override a [DocBookOpenAfter] setting when you want to open an element or hierarchy other than what the setting specifies.

17.5.10 Configuring multi-paragraph list items

By default, at the end of each paragraph **Mif2Go** closes the block element to which the paragraph format is mapped (see §17.4.2 [Mapping paragraph formats to DocBook](#)

elements on page 565). If any list items in your document include multiple paragraphs or sublists, you must make sure that each `<listitem>` can include more than one block element, but also that the last item in each list or sublist does not slurp up any following paragraphs.

To configure list elements:

Map formats to `<para>` instead of to `<listitem>`.

Specify ancestry for each format.

Make each format first in `<listitem>`.

Make sure each list ends where it should.

*Map formats to
<para> instead of
to <listitem>*

Map list-item paragraph formats to `<para>` rather than to `<listitem>`:

```
[DocBookParaTags]
Numbered1 = para
Numbered = para
Bulleted = para
BulletedLast = para
```

*Specify ancestry
for each format*

Designate the appropriate ancestors for each type of list element:

```
[DocBookParents]
Numbered1 = orderedlist listitem
Numbered = orderedlist listitem
Bulleted = itemizedlist listitem
BulletedLast = itemizedlist listitem
```

*Make each format
first in
<listitem>*

Make sure each list-item paragraph is first in its `<listitem>` element:

```
[DocBookFirst]
Numbered1 = orderedlist listitem
Numbered = listitem
Bulleted = listitem
BulletedLast = listitem
```

*Make sure each
list ends where it
should*

If the last paragraph in a multi-paragraph list item is followed by a paragraph whose format is mapped to an element that is valid in `<listitem>`, that paragraph will be included in the list item. To prevent including the following paragraph, you can explicitly close the list:

```
[DocBookCloseAfter]
BulletedLast = itemizedlist listitem
```

Or insert a **DocBookCloseAfter** marker in the last list-item paragraph, with content `itemizedlist orderedlist`.

As an alternative, you can make sure the list closes before the following paragraph:

```
[DocBookCloseBefore]
Body = itemizedlist orderedlist
```

Or insert a **DocBookCloseBefore** marker in the following paragraph, with content:

```
itemizedlist orderedlist
```

17.5.11 Specifying DocBook element levels

Generally speaking, you should not specify element levels unless there really is no other way to properly nest an element; hard-coded levels can cause obscure damage to the output.

To specify the level at which a block element should appear in DocBook output, you can assign a level number to any FrameMaker paragraph formats that are mapped to the element (see §17.4.2 [Mapping paragraph formats to DocBook elements](#) on page 565). However, for most nesting issues, you should use settings that specify ancestry rather than

level; see §17.5.2 [Designating DocBook ancestor elements](#) on page 573. Assign levels only for the following purposes:

- to identify paragraph formats mapped to `<title>` that should start new topics; assign level 1 to each such format
- to handle unusual situations that cannot be addressed any other way.

To specify the level of a DocBook block element:

```
[DocBookLevels]
; Frame para format (wildcards OK) = level in DocBook (not Frame) file
; required for the DocBookParaTag specified for this element.
FmtName = N
```

The lower the level number, the higher the level; `<set>` is level 0, the root. You cannot put anything else at level 0. The set title is at level 1. The first book title in the set is at level 2 (a title below `<set>` and `<book>`).

For example:

```
[DocBookLevels]
Title=1
GlossItem=1
Heading1=3
DefTerm=5
ParamTerm=5
```

In this example the element levels would be `<body>` = 1, `<section>` = 2, the title under `<topic>` (mapped implicitly from paragraph format *Title*) = 1, and any title under `<section>` (mapped explicitly from a *Heading1* format) = 3. *GlossItem* is assigned level 1 because this format is mapped to `<glossterm>`, which is the first element in a glossary topic (equivalent to `<title>` in other topic types).

To override the assigned level of a particular paragraph, place a **DocBookLevel** marker in the paragraph. A **DocBookLevel** marker specifies the level at which the current block element should appear in the DocBook file, overriding whatever is specified for the format in `[DocBookLevels]`. The content of a **DocBookLevel** marker is a single integer.

17.6 Designating ancestors for table elements

To specify the ancestor elements **Mif2Go** must use for `<table>` elements:

```
[DocBookOptions]
; TableParents = parents for table tags, default none (use content
; model), may include sets from [DocBookElementSets].
TableParents =
```

List ancestors in hierarchical order; see §17.5.2 [Designating DocBook ancestor elements](#) on page 573. You can include element sets, as well as single elements; see §17.5.5 [Specifying alternate ancestries for the same element](#) on page 575. If you do not specify any ancestor elements, **Mif2Go** picks the first valid element listed in the content model, which might not be what you had in mind.

To specify ancestry for a single `<table>` element or a discrete group of `<table>` elements, assign the list to the table ID (see §24.2 [Defining sets of tables](#) on page 728). For example:

```
[TableGroup]
FormatA = chart
aa654321 = chart
FormatC = textframe
Unruled = textframe
```

```
[DocBookTableParents]
; table ID (not type) = parents to be used for root table element
chart = section
aa654321 = example
textframe = conbody
```

You can make a single `[DocBookTableParents]` setting in an `HTMConfig` marker, also; see §33.2.2 [Overriding settings with configuration markers](#) on page 921.

17.7 Specifying options for figure elements

In this section:

§17.7.1 [Deciding what to include in a figure element](#) on page 581

§17.7.2 [Specifying ancestry for figure elements](#) on page 581

§17.7.3 [Omitting size attributes from images for DocBook](#) on page 582

17.7.1 Deciding what to include in a figure element

When **Mif2Go** wraps image and title in a `<figure>` element, by default **Mif2Go** closes the `<figure>` element before moving on to the following content. To direct **Mif2Go** to include in `<figure>` any following elements that are valid:

```
[DocBookOptions]
; CloseFigAfterImage = Yes (default)
; or No (leave figure open for more)
CloseFigAfterImage = No
```

By default, **Mif2Go** wraps all contiguous images and their titles in a single `<figure>` element. To make sure each of a series of images is wrapped in its own `<figure>` element:

```
[DocBookOptions]
; MultiImageFigures = Yes (default)
; or No (allow only one image in a figure)
MultiImageFigures = No
```

When an unstructured `FrameMaker` document includes several images in a row with only their titles in between, by default **Mif2Go** assumes that these titles follow their respective images. To specify that figure titles precede their images instead:

```
[DocBookOptions]
; FigureTitleStartsFigure = No (default, title is below image),
; or Yes (title is above image)
FigureTitleStartsFigure = Yes
```

17.7.2 Specifying ancestry for figure elements

To specify the ancestor elements **Mif2Go** must use to wrap `<figure>` elements:

```
[DocBookOptions]
; ImageParents = parents for image tags, default none (use content
; model), may include sets from [DocBookElementSets].
ImageParents = list of parent elements
```

List ancestors in hierarchical order; see §17.5.2 [Designating DocBook ancestor elements](#) on page 573. You can include element sets, as well as single elements; see §17.5.5 [Specifying alternate ancestries for the same element](#) on page 575. If you do not specify any ancestor elements, **Mif2Go** picks the first valid element listed in the content model, which might not be what you had in mind.

For example, suppose you want most of your images wrapped in <section>, except for those that occur in paragraphs that are mapped to <example>:

```
[DocBookOptions]
ImageParents = $iparents

[DocBookElementSets]
$iparents = section example
```

To specify ancestry for a single image element or a discrete group of image elements, assign the parent name or parent set name to the graphic ID of the image (see §5.3 [Identifying files and objects](#) on page 117), or to the graphic group ID (see §23.5.1.4 [Creating named groups of graphics](#) on page 710). For example, to make sure icons in table cells have <entry> as a parent:

```
[GraphGroup]
ab01f853 = alerts
ab012c13 = alerts
ab00b5d3 = alerts

[DocBookImageParents]
; image ID (may be group) = parents to be used for image element.
alerts = entry
```

You can make a single [DocBookImageParents] setting in an HTMLConfig marker, also; see §33.2.2 [Overriding settings with configuration markers](#) on page 921.

*Sequence matters
in element sets*

Although **Mif2Go** knows which elements are valid within other elements, **Mif2Go** has no idea at all about required sequences of elements. For example, if you set:

```
[DocBookElementSets]
$iparents = section entry example graphic
```

Mif2Go will always choose example over graphic. Where the image is valid in both <graphic> and <example>, **Mif2Go** lacks any real criterion for choosing one over the other. Instead, **Mif2Go** selects, from the list of candidates, the first element that is valid as a parent of the image element.

In this example, if more of your images belong in <graphic>, you could set:

```
[DocBookElementSets]
$iparents = section entry graphic example
```

and then use [DocBookImageParents] for the lesser number of images that should be in <example>.

17.7.3 Omitting size attributes from images for DocBook

To eliminate width and height attributes from images for DocBook:

```
[Graphics]
; GraphScale = Yes (default) to put out width and height attributes,
; or No to eliminate them all
GraphScale = No
```

If you do not specify any setting for GraphScale, width and height attributes are included.

17.8 Overriding DocBook settings with markers

You might need to insert markers in your FrameMaker document to override configuration settings for particular DocBook elements. **Mif2Go** provides predefined marker types for this purpose, listed in [Table 17-2](#). Most of these marker types are intended to provide ways to cope with unusual situations. If you have a consistent template, and use normal

FrameMaker cross references and hypertext links, for the most part you can get by with just configuration settings.

Table 17-2 Predefined marker types for DocBook

marker type	Content	Ref.
DocBookAlias	Alternate name for FrameMaker format for the current block	17.4.2.5
DocBookAttribute	Attributes other than ID of a non- <code><xref></code> block element or parent	17.4.5.1
DocBookCloseAfter	Ancestor elements to be closed just after current block element ends	17.5.8.2
DocBookCloseBefore	Ancestor elements to be closed just before current block element starts	17.5.8.1
DocBookCode	XML code to be inserted at the marker location	
DocBookElemID	ID attribute for the current block element	17.4.4.4
DocBookEndElem	Tag name for an inline element, to place at the end of the span	17.4.3.3
DocBookFirst	Ancestor elements under which the current block element must be first	17.5.6
DocBookLevel	Level where the current block element should appear in the DocBook file	17.5.11
DocBookOpenAfter	Elements to be opened just after current block element ends	17.5.9.2
DocBookOpenBefore	Enclosing elements to be opened just before current block element starts	17.5.9.1
DocBookParent	Required ancestors for the current block element	17.5.2
DocBookParentID	ID of the interpolated parent of the current block element	17.4.4.4
DocBookStartElem	Tag name for an inline element, to place at the start of the span	17.4.3.3
DocBookTag	Element name mapping for the current block (not inline) element	17.4.2.4

18 Splitting and extracting files

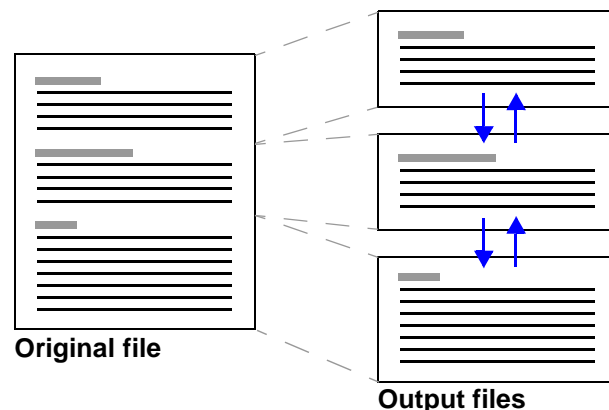
This section shows how **Mif2Go** can divide a FrameMaker file, based on criteria you provide, into smaller files, to convert to HTML; and how **Mif2Go** can extract sections out of the middle of a file, to create excerpts in their own files. Topics include:

- §18.1 [Splitting versus extracting](#) on page 585
- §18.2 [Splitting files](#) on page 586
- §18.3 [Extracting files](#) on page 591
- §18.4 [Identifying split and extract files](#) on page 593
- §18.5 [Inserting HTML code in split and extract files](#) on page 598
- §18.6 [Referencing split and extract files](#) on page 600
- §18.7 [Customizing and replacing extracts](#) on page 601

18.1 Splitting versus extracting

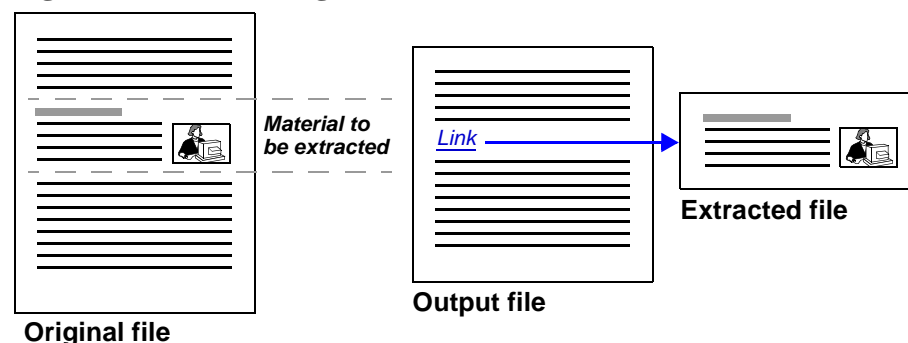
When you *split* a FrameMaker file, as [Figure 18-1](#) shows, each piece of the file becomes a file in its own right, typically addressing a single topic; see §18.2 [Splitting files](#) on page 586.

Figure 18-1 Splitting a file



When you *extract* part of a FrameMaker file, the parent file is converted minus the extracted portion, and the extracted portion becomes a file in its own right. The extracted part is usually replaced in the parent file by a link to the extract file, as [Figure 18-2](#) shows; see §18.3 [Extracting files](#) on page 591.

Figure 18-2 Extracting a file



If you do not split files, you might want some space or a separator in HTML output between the end of one topic and the start of the next.

To insert code before topic-start titles (for example):

```
[Inserts]
TopicBreak = <p class="Body"><br/><br/></p>
```

See §18.5.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 599.

18.2 Splitting files

HTML files are typically much smaller than FrameMaker chapter files. A large HTML file can take an unacceptable amount of time to download from the Web. If you are producing HTML, XML, or HTML-based Help, you will want to split up your FrameMaker files so each topic is in its own file.

If you are converting your FrameMaker index to HTML, or using **Mif2Go** to generate an index for HTML-based Help, **Mif2Go** automatically creates an index entry for each split file when a split occurs within text between index-marker <\$startrange> and <\$endrange> entries.

In this section:

§18.2.1 [Designating split points](#) on page 586

§18.2.2 [Managing split points](#) on page 588

§18.2.3 [Combining instead of splitting files](#) on page 591

18.2.1 Designating split points

You can designate places to split a FrameMaker file based on the occurrence of any of the following:

[Paragraph or character formats in text](#) on page 586

[Paragraph formats in tables](#) on page 587

[Page breaks](#) on page 587

[FrameMaker markers](#) on page 587

[Hypertext alert markers](#) on page 587

Although you can use all these methods in the same FrameMaker document, usually the paragraph-format method alone is adequate. (Do not use any of these methods if you are producing DITA XML output; files are split another way for DITA.)

Note: If you are using the **Mif2Go** plug-in, **Mif2Go** tries to determine automatically the most likely split points; see §1.5 [How Mif2Go works](#) on page 62. **Mif2Go** lists under [HTMLParaStyles] the paragraph formats it chooses for split points. *Be sure you inspect these proposed assignments, and correct any that are inappropriate.* If you are generating HTML-based help, also check the matching list under [HelpContentsLevels].

*Paragraph or
character formats
in text*

You can assign the Split property to a paragraph or character format, so a new HTML page begins wherever an instance of the format appears in your FrameMaker document:

```
[HTMLParaStyles]
; Paragraph format = Split starts a new HTML page at the start of the
; paragraph.
HeadingPara = Split
```

If you use this method to designate split points you will almost certainly want to add the Title property too, maybe others:

```
[HTMLParaStyles]
HeadingPara = Split Title NoFrameBelow
```

Note: In your FrameMaker document, if you separate the *HeadingPara* paragraph from following text with an empty paragraph in the same *HeadingPara* format (perhaps for spacing), you will get *two* split files: one containing only the heading, the other containing only the text. See §18.2.2 [Managing split points](#) on page 588.

*Paragraph
formats in tables*

Normally **Mif2Go** does not allow a file split when a paragraph for which you have specified the *Split* property occurs within a table. However, you can override this prohibition with the following setting:

```
[Tables]
; AllowTbSplit = No (default)
; or Yes (allow file split for head in table)
AllowTbSplit=Yes
```

When *AllowTbSplit=Yes*, the actual split is made at the paragraph containing the table anchor, not at the paragraph in the table.

Page breaks

You can designate whether “hard” page breaks in your FrameMaker document result in a new HTML page, a horizontal bar, or neither:

```
[HTMLOptions]
; PageBreaks = Rule (<hr>), None, or Split
PageBreaks=Split
```

Setting *PageBreaks=Split* does not cause splitting at page breaks that are automatically generated by FrameMaker (in the middle of a paragraph, for example), but only at page breaks specified as part of a paragraph format, either by definition (start at top of page) or as an override.

*FrameMaker
markers*

You can add FrameMaker marker type **Split** to your FrameMaker document, and insert a **Split** marker wherever you want a split point. However, if you insert a **Split** marker in a paragraph whose format is also assigned the *Split* property, the **Split** marker is ignored.

You can also invent a new marker type (for example, **Splitmark**) to use for split points, or you can add the *Split* property to an existing marker:

```
[MarkerTypes]
Splitmark = Split
Subject = Split
```

Note: If you also assign one or more of marker type properties *Cross-Ref*, *Title*, or *FileName*, those properties must be specified *after* the *Split* property; for example:

```
[MarkerTypes]
Filespec = Split FileName
Topicmark = Split Title
```

See §29.4.2 [Observing restrictions on redefining marker behavior](#) on page 840.

A **Split** marker can be placed before the title element in the topic. It must be placed to the left of any **Title**, **FileName**, or **Cross-Ref** marker that occurs in the same paragraph.

See §29 [Working with FrameMaker markers](#) on page 831 for more information.

*Hypertext alert
markers*

You can mark a split point in your document by inserting a hypertext **alert** marker that contains only the word “split”. See §34.1.2 [Using markers to add links and instructions](#) on page 935.

18.2.2 Managing split points

You can selectively disable splitting to keep content together, such as at the start of a FrameMaker file, or when a single heading would be the only content.

In this section:

- §18.2.2.1 [Preventing splits that leave empty files](#) on page 588
- §18.2.2.2 [Preventing splits that create unwanted files](#) on page 588
- §18.2.2.3 [Preventing splits that leave dangling headings](#) on page 589
- §18.2.2.4 [Keeping headings together when other content intervenes](#) on page 590
- §18.2.2.5 [Specifying split points for multiple-paragraph headings](#) on page 590

18.2.2.1 Preventing splits that leave empty files

Often the first split point is at the very first paragraph in a FrameMaker file, which results in an empty first HTML file. You could discard this file when you package output for production; however, doing so would mean that you might not be able to use navigation macros (see §20.4 [Creating a browse sequence](#) on page 635). Unless the HTML output is destined for an automated system that requires renaming files, a better approach is to prevent the split.

To suppress the first split if it would result in an empty file, set `StartingSplit=No`:

```
[HTMLOptions]
; StartingSplit = Yes (default, allows split at start of file) or No
StartingSplit = No
```

When `StartingSplit=Yes`, the first HTML file (which is named for the FrameMaker parent file) cannot be given a title based on a split-point paragraph format; see §18.4.2 [Specifying page titles for split or extract files](#) on page 594. If you discard empty files, **Mif2Go** removes the unwanted empty files from the wrap directory and omits them from the navigation sequence; however, the empty files remain in the project folder, where they still serve a purpose.

When `StartingSplit=No`, you can use a **FileName** marker to rename the first “real” split file to the FrameMaker chapter name with extension `.htm`, forcing this file to overwrite the empty file; see §34.8.3 [Using custom markers to name output files](#) on page 947. This method allows page titles based on split-point paragraphs, and does not interfere with navigation macros.

To avoid creating empty files, we recommend:

```
[HTMLOptions]
StartingSplit = No
```

If the first split is *not* at the first paragraph in a FrameMaker file, see §18.2.2.2 [Preventing splits that create unwanted files](#) on page 588.

18.2.2.2 Preventing splits that create unwanted files

Suppose each of your FrameMaker chapters begins with a paragraph tagged *ChapHead*, followed by a *Heading1* paragraph. Suppose you use *Heading1* as a split point, and the *ChapHead* paragraph merely provides formatting or decoration on the chapter title page. The output will include an HTML file named for the chapter file; and that non-empty file will be included in any browse sequence.

To eliminate the unwanted paragraphs, you have the following choices:

- [Assign both Split and Delete](#)
- [Apply a condition in FrameMaker.](#)

Assign both Split and Delete

Assigning format property *Delete* to *ChapHead* would eliminate paragraph content; also assigning format property *Split* would allow you to use *SmartSplit*:

```
[HTMLParaStyles]
ChapHead = Split Delete
Heading1 = Split
```

You must set *SmartSplit=Yes* and assign appropriate level numbers, as described in §18.2.2.3 [Preventing splits that leave dangling headings](#) on page 589, to get rid of the unwanted file. You will not be able to give the first split file a name different from the FrameMaker chapter file.

Apply a condition in FrameMaker

Instead, you could use conditional text in FrameMaker to mark *ChapHead* paragraphs for hiding. For example, apply a condition named *HeadStuff* to each *ChapHead* paragraph, and in your configuration file specify the following:

```
[Setup]
SetFrameConditions = Yes

[ConditionsShown]
HeadStuff = No
```

See §5.4.1 [Applying condition Show/Hide settings](#) on page 123.

See also:

§18.6 [Referencing split and extract files](#) on page 600

§21.3.12 [Eliminating unwanted paragraphs](#) on page 652

18.2.2.3 Preventing splits that leave dangling headings

Suppose you have designated the following criterion for split points:

```
[HTMLParaStyles]
Heading2 = Split
```

Now suppose your FrameMaker document has content organized as follows:

```
Heading1
  Heading2
    Body text
  Heading2
    Body text
Heading1
. . .
```

And you want split points as follows:

```
Heading1
  Heading2
    Body text
----- split here
  Heading2
    Body text
----- split here
Heading1
. . .
```

That is, you want to suppress a split between the first *Heading1* and the first *Heading2*.

Instead of the first *Heading2* you could use a different paragraph format that is not assigned the *Split* property, such as *Heading2First*, and add that format to the paragraph catalog. Or, you could use `[HelpContentsLevels]` to assign level numbers to your heading formats, and set *SmartSplit=Yes*:

```
[HelpContentsLevels]
; FM paragraph format name = TOC level (MS or Java)
```



```

Heading1 = 1
Heading2 = 2

[HTMLParaStyles]
Heading1 = Split
Heading2 = Split

[HTMLOptions]
StartingSplit=No
; SmartSplit = No (default) or Yes (suppress splits in sequences of
; heads which lack text bodies, as long as the [HelpContentsLevels]
; values continue to increase)
SmartSplit = Yes

```

You need `StartingSplit=No` to avoid creating a blank file; see §18.2.2.1 [Preventing splits that leave empty files](#) on page 588.

The effect of these settings is to suppress splits as long as the heading level number increases for each paragraph in an unbroken sequence of paragraphs. For example, these settings would keep all of the following together:

```

Heading1
  Heading2
    Heading3
      Body text

```

In other words, `SmartSplit` prevents splits when both of the following are true:

- the later heading is subordinate to the earlier heading
- there is no body content between headings (but see §18.2.2.4 [Keeping headings together when other content intervenes](#) on page 590).

Note: It is not a good idea to use `SmartSplit` in conjunction with trails of links; see §20.2.1 [Understanding trails of links](#) on page 627.

Note: It is not a good idea to use `SmartSplit` in conjunction with local TOCs; see §20.3 [Including local TOCs](#) on page 631.

18.2.2.4 Keeping headings together when other content intervenes

To keep sections from splitting into separate HTML files even when there is body content between headings that are assigned the `Split` property, you have two choices:

[Prevent the content from affecting the split](#)

[Use alternative heading formats.](#)

*Prevent the
content from
affecting the split*

To prevent a paragraph format (for example, *ThinLine*) that occurs between headings from interfering with keeping the headings together when `SmartSplit=Yes`:

```

[HTMLParaStyles]
; NoSplit prevents the para format from interfering with SmartSplit
; when it occurs between heads that would otherwise be kept together.
ThinLine = NoSplit

```

*Use alternative
heading formats*

For the affected headings, consider using alternate paragraph formats with the same properties. For example, you could define a paragraph format named *Heading1x* and use that format instead of *Heading1* wherever you want to prevent a *Heading1* split.

18.2.2.5 Specifying split points for multiple-paragraph headings

Suppose the main headings in your FrameMaker document use two paragraph formats, one for the text of the heading, and another for the heading number; for example, *ChapterTitle* and *ChapterNumber*. You might assume that splitting files on *ChapterNumber* would be the right thing to do, and usually that would be true:

```
[HTMLParaStyles]
ChapterNumber=Title Split

[HelpContentsLevels]
ChapterTitle=1
```

If *ChapterNumber* precedes *ChapterTitle* in your FrameMaker document, **Mif2Go** splits the file at *ChapterNumber*, and the text of *ChapterTitle* becomes the reference to that file.

However, if *ChapterNumber* follows *ChapterTitle*, as it does in some designs, the text of *ChapterTitle* becomes a reference to the *previous* split file, and appears at the end of that file. To correct this problem you would specify *ChapterTitle* instead of *ChapterNumber* for the split point:

```
[HTMLParaStyles]
ChapterTitle=Title Split

[HelpContentsLevels]
ChapterTitle=1
```

18.2.3 Combining instead of splitting files

To convert a FrameMaker book to a single HTML output file instead of splitting the chapters into smaller topics, you must start with a single FrameMaker file. Create a new FrameMaker file, and import into the new file, *by reference*, each of the chapter files, in the same order they appear in the book. Each chapter file becomes a text inset in the new file. You never edit this file again; just update it after you make changes in your FrameMaker book, then use **Mif2Go** to export the new file to HTML.

We do not recommend this method, even for small projects, because it produces an HTML page that is very slow to load.

For HTML, the best practice is to reduce the size of each page to one topic, preferably a topic that does not require any scrolling to see the whole page. When people scan for information, they might go from page to page without looking past the first screen of each.

18.3 Extracting files

You can direct **Mif2Go** to extract part of a document into an *extract file*, optionally replacing the extracted portion in the original document with a link to the extract file, as shown in [Figure 18-2](#) on page 585. You can use this feature to move material such as graphics and stepwise procedures into their own files.

In this section:

§18.3.1 [Enabling and disabling extract processing](#) on page 591

§18.3.2 [Delimiting material to extract](#) on page 592

18.3.1 Enabling and disabling extract processing

To enable extracts for your entire document, specify the following setting:

```
[HTMLOptions]
; ExtractEnable = Yes (allow extract files) or No (default, disable)
ExtractEnable=Yes
```

To turn extract processing on and off for some parts of your document, you can assign extract switch properties `ExtrEnable` and `ExtrDisable` to paragraph formats. For example, you might define paragraph formats *ExEnable* and *ExDisable*, and use them to delineate the portions of your document where extract processing should occur:

```
[HTMLParaStyles]
; ExtrEnable and ExtrDisable turn extract processing on or off; the
; starting state is given in [HTMLOptions]ExtractEnable=Yes or No
ExEnable=ExtrEnable Delete
ExDisable=ExtrDisable Delete
```

Once you have placed such paragraphs in your document, you cannot turn off all extract processing just by setting [HTMLOptions]ExtractEnable=No; instead you must delete all ExtrEnable assignments in [HTMLParaStyles].

18.3.2 Delimiting material to extract

To mark the start and end points of material to be extracted, you can assign properties to paragraph formats, or you can use markers. For either of these methods to take effect, you must also enable extract processing; see §18.3.1 [Enabling and disabling extract processing](#) on page 591.

In this section:

§18.3.2.1 [Using existing paragraph formats to delimit extracts](#) on page 592

§18.3.2.2 [Creating special paragraph formats to delimit extracts](#) on page 592

§18.3.2.3 [Using markers to delimit extracts](#) on page 593

See also:

§18.3.1 [Enabling and disabling extract processing](#) on page 591

18.3.2.1 Using existing paragraph formats to delimit extracts

To use paragraph formats that are already in place to mark the start and end points of material to extract, assign [HTMLParaStyles] extract properties to those paragraph formats. For example:

```
[HTMLParaStyles]
; doc format (para or char) = keywords for functions and properties
; ExtrStart begins an extract.
; ExtrFinish is the last part of the extract.
; ExtrEnd ends the extract, but is not part of the extract itself
FigureTitle=ExtrStart
zFigAnchor=ExtrFinish
```

To extract a single paragraph, assign both ExtrStart and ExtrFinish to the paragraph format.

ExtrFinish and ExtrEnd are alternate ways to end an extract. You do not have to use both, but doing so is harmless. If you do not use one or the other, the extract ends gracefully at the next split point, or at the end of the file.

Note: For any of these settings to take effect, you must enable extracts; see §18.3.1 [Enabling and disabling extract processing](#) on page 591.

For examples, see §18.7.4 [Specifying extracts: an example](#) on page 607.

18.3.2.2 Creating special paragraph formats to delimit extracts

You can create special paragraph formats to mark the boundaries of extracts, as follows:

1. Invent two new paragraph formats; for example, *ExStart* and *ExEnd*.
2. Insert an element with `outputclass=ExStart` just before material to be extracted.
3. Put an *ExStart* paragraph just before material to be extracted.
4. Insert an element with `outputclass=ExEnd` just after material to be extracted.

5. Put an *ExEnd* paragraph just after material to be extracted.
6. Make the *ExStart* and *ExEnd* paragraphs conditional for on-line use, so they do not appear in your regular print files.
7. Assign the following properties to these special paragraph formats:

```
[HTMLParaStyles]
ExStart=ExtrStart Delete
ExEnd=ExtrEnd Delete
```

The `Delete` property excludes the *ExStart* and *ExEnd* paragraphs from text in the HTML file. In effect, these special paragraphs serve only as directives to **Mif2Go**. If you omit the *ExEnd* paragraph, the extract ends at the next split point, or at the end of the file.

Note: For these settings to take effect, you must enable extracts; see §18.3.1 [Enabling and disabling extract processing](#) on page 591.

18.3.2.3 Using markers to delimit extracts

You can specify the boundaries of an extract with carefully placed markers. **Mif2Go** ignores the content of these markers, so you can use the content for comments:

ExtrStart	First part of an extract; insert in the first paragraph in the extract, to the left of any Title , FileName , or Cross-Ref marker that occurs in the same paragraph.
ExtrFinish	Last part of an extract; insert in the last paragraph of the extract.
ExtrEnd	End of an extract; insert after the last paragraph of the extract, and before the end of the next paragraph.

For easy maintenance, the best place for these markers is at the start of a paragraph.

To extract a single paragraph, insert both **ExtrStart** and **ExtrFinish** markers in the paragraph; the **ExtrStart** marker must precede the **ExtrFinish** marker.

ExtrFinish and **ExtrEnd** markers are alternate ways to end an extract. You do not have to use both, but doing so is harmless. If you do not use one or the other, the extract ends gracefully at the next split point, or at the end of the file.

Note: For any of these markers to take effect, you must enable extracts; see §18.3.1 [Enabling and disabling extract processing](#) on page 591.

18.4 Identifying split and extract files

You can specify titles and meta information for split and extract files. **Mif2Go** names the files, and retains those names for internal purposes; it is best not to try to rename split or extract files.

In this section:

§18.4.1 [Understanding how split and extract files are named](#) on page 593

§18.4.2 [Specifying page titles for split or extract files](#) on page 594

§18.4.3 [Supplying <meta> text for split or extract files](#) on page 598

18.4.1 Understanding how split and extract files are named

When **Mif2Go** splits a FrameMaker file, the result is a series of HTML files. The first HTML file contains whatever was in the FrameMaker file before the first split point; this file keeps the name of the FrameMaker file, but with extension `.htm`. *Do not rename this file*, because it is required for subsequent conversions. **Mif2Go** names the rest of the split

files, and all extracted files, as described in §5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117.

In this section:

§18.4.1.1 [Keeping split and extract file names unique](#) on page 594

§18.4.1.2 [Replacing split and extract file names](#) on page 594

18.4.1.1 Keeping split and extract file names unique

To determine in advance what ObjectID will be used to name a split or extract file, click the heading that will start the file. Then, without moving the mouse, Shift-click. The paragraph's decimal ObjectID appears on the status line as the FrameMaker ParaID.

Because the name of a split or extract file depends only on the FileID of the parent file and the ObjectID of the paragraph at the split or extract point, the file name is not likely to change, even if you add more split points earlier in the file.

However, the paragraph ObjectID *will* change if you do any of the following (see §5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118):

- copy and paste the paragraph
- delete the paragraph and then retype it
- hide the paragraph with a condition and then show it.

Just editing a paragraph does not affect its ObjectID. See §5.3 [Identifying files and objects](#) on page 117.

18.4.1.2 Replacing split and extract file names

If your HTML output will be used in an automated system with file-name requirements that conflict with **Mif2Go**-assigned file names, see §34.8 [Renaming output files for automated systems](#) on page 946.

When you use **Mif2Go**-provided options to automatically gather and package HTML files for production (see §35 [Producing deliverable results](#) on page 955), all the HTML files **Mif2Go** creates in the course of a conversion are included in the production package. This means that the very first HTML file for each FrameMaker file (the file named for the parent file) will be included, even if it is an empty file. For ways to eliminate these empty files, see §18.2.2.2 [Preventing splits that create unwanted files](#) on page 588.

18.4.2 Specifying page titles for split or extract files

You can specify the HTML page <title> values for HTML output files in any of the following ways:

- via [HTMLOptions]Title = *My default title for all files*
- via [Titles]filename = *My title for this file*
- via [HTMLParaStyles]first paragraph format = Title
- via FrameMaker marker **Title**

You can use more than one method, and you can use macros and macro variables with any of these methods.

In this section:

§18.4.2.1 [Understanding title assignment precedence](#) on page 595

§18.4.2.2 [Assigning a title with a paragraph format](#) on page 595

§18.4.2.3 [Specifying a title prefix or suffix](#) on page 596

- §18.4.2.4 [Assigning a title with a file name](#) on page 596
- §18.4.2.5 [Assigning a title with a marker](#) on page 597
- §18.4.2.6 [Assigning a default title](#) on page 597
- §18.4.2.7 [Assigning a computed title](#) on page 597

18.4.2.1 Understanding title assignment precedence

You can specify the HTML page title for a particular output file more than one way. For example, in general you might want to use the paragraph-format method (which applies to all files), then override that method with a **Title** marker or a [Titles] assignment for selected files. [Table 18-1](#) shows which method takes precedence if more than one method applies to a given output file.

Table 18-1 Precedence of HTML page titles

Precedence	Method	Comments
Highest	[Titles]	Titles assigned in this section cannot be overridden
Intermediate	[HTMLParaStyles] or Title , Config , or HTMConfig marker	If you use both methods for a given split or extract part, whichever occurs first in the source file (the Title , Config , or HTMConfig marker or an instance of the Title paragraph format) takes precedence
Lowest	[HTMLOptions]	Any other method overrides an [HTMLOptions]Title value

Default title If you do not specify any page titles, the default title for each output file is *Test File from Mif2Go*; any other title specification overrides this default. If some of your output files show *Test File from Mif2Go* as the title, this means you did not manage to specify titles for those files. See §18.4.2.6 [Assigning a default title](#) on page 597.

18.4.2.2 Assigning a title with a paragraph format

For the page title of a split or extract file, you can use the content of the heading that caused the split or extract, or the content of the first instance of any other paragraph in the split or extracted part. Optionally, you can also specify a prefix or suffix or both; see §18.4.2.3 [Specifying a title prefix or suffix](#) on page 596.

Heading content as title To use the content of a heading as a page title, assign the **Title** property to the heading paragraph format; for example:

```
[HTMLParaStyles]
; Title uses head as HTML page title, see [Titles]; may be preceded
; by [StyleTitlePrefix] and followed by [StyleTitleSuffix]
FigCaption=ExtrStart Title
Heading2=Split Title
```

Headings in tables Normally **Mif2Go** does not create a page title if the heading for which [HTMLParaStyles] has the **Title** property is within a table. If you need titles from headings in tables (for example, if your chapter headings are in single-cell tables) specify the following setting:

```
[Tables]
; AllowTbTitle = No (default) or Yes (allow title from head in table)
AllowTbTitle=Yes
```

Otherwise, an HTML page title will not be created from such a heading.

Macros and macro variables If you use macros or macro variables in the text of a paragraph to which you have assigned the **Title** property, you must also assign property **Raw**; otherwise, the characters <, >, and

& all get turned into HTML entities; and instead of being expanded, a macro appears literally in the output. See §21.3.6 [Stripping paragraph properties](#) on page 650.

See also:

§18.7.2 [Customizing title text for extracts](#) on page 602

§21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646

18.4.2.3 Specifying a title prefix or suffix

If you assign titles with paragraph formats (see §18.4.2.2 [Assigning a title with a paragraph format](#) on page 595), you can also specify a prefix, a suffix, or both as part of the title. For example:

```
[HTMLParaStyles]
Heading2=Split Title

[StyleTitlePrefix]
; doc style = prefix to use (if any) for file title in para content
Heading2=Course IV:

[StyleTitleSuffix]
; doc style = suffix to use (if any) for file title in para content
Heading2= (draft 1)
```

With these settings, a *Heading2* paragraph that has content *Prerequisites* would result in a split file with the following <title> element:

```
<title>Course IV: Prerequisites (draft 1)</title>
```

*Excluded from
trails and local
TOCs*

Any title prefix or suffix you specify with these settings is excluded from the title when it appears in a trail of links (see §20.2 [Generating trails of links](#) on page 627) or in a local TOC (see §20.3 [Including local TOCs](#) on page 631).

Trailing spaces

Trailing spaces you type at the end of the title prefix setting are included in the prefix.

Leading spaces

If you type one or more leading spaces after the equals sign at the beginning of the title suffix text, **Mif2Go** removes exactly one of them (see §4.4 [Understanding the rules for configuration settings](#) on page 102). If you want a single leading space for the title suffix, supply exactly two spaces after the equals sign.

*Macros and
macro variables*

You can use macros and macro variables (see §28.1 [Defining and invoking macros](#) on page 787) in sections [StyleTitlePrefix] and [StyleTitleSuffix]. For example, suppose you have defined a FrameMaker variable called *CourseNum* in your document, and you want to use the value of that variable as a title prefix:

```
[StyleTitlePrefix]
Heading2=<$$CourseNum>:
```

This works because FrameMaker variables can be employed as **Mif2Go** user variables; see §28.3.1 [Creating and invoking macro variables](#) on page 796.

18.4.2.4 Assigning a title with a file name

You can assign title text to the name of a split or extract file, to specify a title explicitly; this assignment takes precedence over a title for the same file specified with a format. You must assign the title text to the internal file name assigned by **Mif2Go** (see §18.4.1 [Understanding how split and extract files are named](#) on page 593), not to any replacement name you may have specified for a split or extract file. For example:

```
[Titles]
; html filename = title, overrides [HTMLParaStyles] Title setting
; for split or extract files, use FileID+UID, such as mr516387
af345674=HTML Help, JavaHelp, and Oracle Help for Java
ba134256=Export dialog
```


**Macros and
macro variables**

You can use macros and macro variables in the [Titles] section. For example, suppose you want to use as a title the name of the FrameMaker file from which the HTML file was generated:

```
[Titles]
*=<$$_basefile>
```

This setting would provide the base name (without extension) of the FrameMaker source file as the title of each HTML output file. However, a more efficient way to do the same thing would be to assign macro variable <\$\$_basefile> to the Title keyword; see §18.4.2.7 [Assigning a computed title](#) on page 597.

See also:

§4.6 [Using wildcards in configuration settings](#) on page 106

§28.3.4 [Using predefined macro variables](#) on page 800

18.4.2.5 Assigning a title with a marker

You can insert a FrameMaker marker of type **Title** in the first paragraph of a split or extract, and supply the text of the title as the marker content. To use this method you must add custom marker type **Title** to your FrameMaker document. The content of the marker becomes the page title of the split or extract file. You can use macros and macro variables in the marker content.

As an alternative, you can add the Title property to a different marker type: for example, custom marker type **Split** (see §18.2.1 [Designating split points](#) on page 586), or custom marker type **ExtrStart**; and specify the title as the content. See §29 [Working with FrameMaker markers](#) on page 831 for more information.

18.4.2.6 Assigning a default title

To specify a default title for any otherwise untitled HTML page:

```
[HTMLOptions]
;Title = default title for HTML files,
; overridden by all other settings
Title=My default page title
```

Titles specified any other way override the value assigned to Title.

Note: If you do not specify a value for Title, the title of any otherwise untitled HTML page in your output is *Test File from Mif2Go*.

18.4.2.7 Assigning a computed title

If titles for your HTML pages can be determined based on the value of a macro variable, or on values obtained by expanding a macro, you can assign the macro or macro variable to the [HTMLOptions]Title keyword.

For example, to provide a page title that consists of the FrameMaker source file name followed by an integer that increments for each HTML file generated:

```
[HTMLOptions]
Title=<$$PageTitle>

[PageTitle]
<$$_basefile><$$PgNumber++ as %-0.3d>

[MacroVariables]
PgNumber=0
```

Any other title specification overrides a computed [HTMLOptions]Title value.

See also:

§28.1 [Defining and invoking macros](#) on page 787

§28.3.3 [Incrementing and decrementing macro variables](#) on page 799

§28.6.3 [Displaying expression results in output](#) on page 813

18.4.3 Supplying <meta> text for split or extract files

To supply text for the <meta> tag content attribute in the head section of each split or extract file, assign the Meta property to a paragraph format. For example:

```
[HTMLParaStyles]
; Meta uses the contents of the para as the content attribute of
; a meta tag in the head.
Metakeys=Meta
```

To supply the name attribute of the meta tag, assign the name to the same format:

```
[StyleMetaName]
; doc style = name to use for meta tag whose content is the para text
Metakeys=keywords
```

See §13.4.6 [Supplying content for the <meta> tag](#) on page 434 for more information.

18.5 Inserting HTML code in split and extract files

Split and extract files require the usual HTML <head> and <body> sections. **Mif2Go** provides code for these sections, just as for any other HTML output file. You can specify additional HTML code, including macros, for **Mif2Go** to insert at several points in these HTML sections.

In this section:

§18.5.1 [Choosing how to insert code in extracts](#) on page 598

§18.5.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 599

§18.5.3 [Using special sections to insert code in extracts](#) on page 600

18.5.1 Choosing how to insert code in extracts

You can specify code to be inserted in extract files by any of the methods listed in [Table 18-2](#). Code can be placed at any of the following locations in an extract file:

- Within the <head> element, after the <title> element
- At the beginning of the <body> element
- Just before the end of the <body> element.

Table 18-2 Extract code insertion methods

Precedence	Type	Code insertion method	Ref.
Highest	Marker	Make code the text (maximum 256 characters) of a marker placed anywhere in the extract	18.7.1
Intermediate	Configuration section	Assign code to the paragraph format designated to start the extract (see 18.3.2.1)	18.5.3
Lowest	Configuration keyword	Assign code to an [Inserts] keyword	18.5.2

The methods in [Table 18-2](#) are listed in order of precedence. For example, if you use both an **ExtrHead** marker containing HTML code and supply an [Inserts]ExtrHead entry specifying HTML code, **Mif2Go** inserts the marker code in the <head> element of the

extract and ignores the `[Inserts]ExtrHead` entry. This allows you to override code specified in the configuration file for any extracts that require different code.

18.5.2 Assigning code to `[Inserts]` keywords for splits and extracts

To specify a predetermined location for HTML code, you can assign the code to a keyword in the `[Inserts]` section. For example:

```
[Inserts]
Top=<$TopNavTable><br />
Bottom=<br /><$BtmNavTable>
```

Table 18-3 lists the basic `[Inserts]` location keywords, and for each keyword describes where the HTML code assigned to that keyword would be invoked in an output file.

Table 18-3 Basic macro-insertion keywords and locations

<code>[Inserts]</code> keyword	Location of macro in HTML file
TopicBreak	Before the <code><title></code> element (between topics) when files are not split
Head	Within the <code><head></code> element, after the <code><title></code> element
HeadEnd	At the very end of the <code><head></code> element
Frames	Between <code><head></code> and <code><body></code> elements, for framesets (split files only)
Top	At the beginning of the <code><body></code> element.
Bottom	Just before the end of the <code><body></code> element.
End	After the end of the <code><body></code> element (to close <code>noframes</code> ; split files only)

`[Inserts]` file-type prefixes

Table 18-4 lists prefixes for the basic keywords (except `TopicBreak`), and for each prefix describes the type of file where HTML code assigned to a keyword with that prefix would be invoked.

Table 18-4 Keyword prefixes for split or extract code insertion

<code>[Inserts]</code> keyword prefix	Type of split or extract file where applicable
First	First split part
Split	Split parts between the first and the last
Last	Last split part
Nonsplit	Files that are not split (among files that are split); use only with <code>Top</code> or <code>Bottom</code>
Extr	Extract file; use only with <code>Head</code> , <code>Top</code> , or <code>Bottom</code>

`[Inserts]` location/file-type variants

Table 18-5 combines prefixes and keywords to show all the keyword variants you can use in the `[Inserts]` section, by type of file and by location within a file.

Table 18-5 Code insertion keywords for split and extract files

Solitary file	First* split file	Intermediate split files	Last** split file	Non-split file (among splits)	Extract file
Head	FirstHead	SplitHead	LastHead	---	ExtrHead
HeadEnd	FirstHeadEnd	SplitHeadEnd	LastHeadEnd	---	ExtrHeadEnd
Frames	FirstFrames	SplitFrames	LastFrames	---	---
Top	FirstTop	SplitTop	LastTop	NonsplitTop	ExtrTop
Bottom	FirstBottom	SplitBottom	LastBottom	NonsplitBottom	ExtrBottom
End	FirstEnd	SplitEnd	LastEnd	---	---

* If FirstBottom is not defined, SplitBottom is used. If SplitBottom is not defined, Bottom is used. If any others are not defined, the corresponding non-First form is used.

** If any of the first three is not defined, the corresponding Split form is used; otherwise the non-Last form (as for the last two).

18.5.3 Using special sections to insert code in extracts

You can assign HTML code (including macros) to the [HTMLParaStyles]ExtrStart format or **ExtrStart** marker (whichever you used to designate the start of the extract) to be invoked in the following sections:

```
[ExtrHead]
; starting format = HTML code for the head of the extract file

[ExtrTop]
; starting format = HTML code for the top of the extract body

[ExtrBottom]
; starting format = HTML code for the bottom of the extract body
```

These sections correspond to [Inserts] keywords ExtrHead, ExtrTop, and Extrbottom; and to marker types **ExtrHead**, **ExtrTop**, and **Extrbottom**; and are used for the same purposes. A marker type overrides a [Extr*] section of the same name, and an [Extr*] section overrides an [Inserts] keyword of the same name; see §18.5.1 [Choosing how to insert code in extracts](#) on page 598.

For examples, see §18.7.4 [Specifying extracts: an example](#) on page 607.

18.6 Referencing split and extract files

You can use the predefined macro variables listed in [Table 18-6](#) to refer to file names and titles of split and extract files. You can use these variables anywhere in a macro, including within JavaScript sections. They are valid in all parts of a file, including within the base part from which the other parts are split or extracted.

For navigation links between FrameMaker files, use Mif2Go-supplied navigation macros instead; see §18.5 [Inserting HTML code in split and extract files](#) on page 598.

Table 18-6 Predefined macro variables for splits and extracts

Type	Variable	Description
File name	<code>\$_basefile</code>	Base name only of the parent file, without extension.
	<code>\$_currbase</code>	Base name only of the current split part, without extension.
	<code>\$_currfile</code>	Current split part: file name with extension
	<code>\$_currfilepath</code>	Current split part: full path and file name with extension
	<code>\$_extrgraphid</code>	Internal file name of first graphic referenced in an extract
	<code>\$_extrfile</code>	Current extract file name
	<code>\$_extrgraph</code>	File name of first graphic in an extract, as modified by any [Graphics]ExtrGraphSuffix; use for thumbnails (see §18.7.3 Replacing extracts with links in the parent file on page 603)
	<code>\$_extrgraphid</code>	Mif2Go internal name of first graphic in an extract
	<code>\$_nextfile</code>	Split part that follows <code>\$_currfile</code>
	<code>\$_prevfile</code>	Split part that precedes <code>\$_currfile</code>
Title	<code>\$_currtitle</code>	Current file title, unaffected by extracts, so it can be used in an extract to get the parent file title.
	<code>\$_basetitle</code>	Original document title, unaffected by splits.
	<code>\$_prevtitle</code>	Title of <code>\$_prevfile</code> split part.
	<code>\$_nexttitle</code>	Title of <code>\$_nextfile</code> split part.
	<code>\$_extrtitle</code>	Current extracted-part title; used in a replacement macro for the extract, to reference the extract title.; see §18.7.3 Replacing extracts with links in the parent file on page 603.
Boolean	<code>\$_firstfile</code>	1 if this is the first split part, otherwise 0; intended for JavaScript use.
	<code>\$_lastfile</code>	1 if this is the last split part; otherwise 0; intended for JavaScript use.

Interfile links

When the current split part is the first or the last in a FrameMaker file, the following macro variables can also apply to the preceding or following FrameMaker file when you use **Mif2Go** navigation macros

```
$_prevfile
$_prevtitle
$_nextfile
$_nexttitle
```

Note: Predefined macro variables `$_firstsfile` and `$_prevsfile` are no longer needed, and their use is deprecated; however, existing deployments are not affected, so you can continue to use these variables in macro code.

See §18.5 [Inserting HTML code in split and extract files](#) on page 598 for additional settings you can use for links between FrameMaker files.

18.7 Customizing and replacing extracts

In this section:

- §18.7.1 [Using markers for extract processing](#) on page 602
- §18.7.2 [Customizing title text for extracts](#) on page 602
- §18.7.3 [Replacing extracts with links in the parent file](#) on page 603
- §18.7.4 [Specifying extracts: an example](#) on page 607

18.7.1 Using markers for extract processing

You can use predefined marker types to supply most extract-file properties. [Table 18-7](#) lists the predefined marker types for extracts.

Table 18-7 Predefined marker types for extracts

Marker type	Purpose	Content use	Placement	Ref.
ExtrBottom	Insert HTML code	Last item in extract <body>	Anywhere in the extract*	18.5.1
ExtrDisable	Turn off extract processing	Ignored	After all or a group of extracts	18.3.1
ExtrEnable	Turn on extract processing	Ignored	Before all or a group of extracts	18.3.1
ExtrEnd	End an extract;	Ignored	After end of last paragraph in extract, and before end of following paragraph	18.3.2
ExtrFinish	Mark last part of extract	Ignored	In the last paragraph of the extract	18.3.2
ExtrHead	Insert HTML code	Placed in extract <head>	Anywhere in the extract*	18.5.1
ExtrReplace	Insert HTML code	Replaces extract in parent file	Anywhere in the extract*	18.7.3
ExtrStart	Begins extract	Ignored	In the first paragraph of the extract; can be used in [Extr*] sections	18.3.2 , 18.5.3
ExtrTop	Insert HTML code	First item in extract <body>	Anywhere in the extract*	18.5.1

* To avoid maintenance headaches, pick a consistent location, such as at the start of the first paragraph.

You can also use any of the **Extr*** marker-type names listed in [Table 18-7](#) as properties of other markers; see §29 [Working with FrameMaker markers](#) on page 831 for more information.

Mif2Go processes every marker that has the same name as an [HTMLParaStyles]Extr* property the same way as that property. Properties assigned via markers take precedence over the same properties assigned via formats; see §18.5.1 [Choosing how to insert code in extracts](#) on page 598.

18.7.2 Customizing title text for extracts

In the [ExtrTitle] section you can assign text for the extract title to the ExtrStart format or **ExtrStart** marker, whichever you used to designate the start of the extract:

```
[ExtrTitle]
; doc format = text for title of the extract file, which may use
; macros, including <$$_currtitle> which has the title of the parent
; file. This is ignored if the Title is set for any para in the
; extract file.
ExtractStartPara=Title for extract file
```

Predefined macro variable <\$\$_currtitle> is the <title> string of the file that originally contained the extracted text; see §18.6 [Referencing split and extract files](#) on page 600.

For example, suppose your FrameMaker document contains a paragraph format that always marks the start of an extract; for example *ProcedureStart*; and suppose you want

each such extract to have a title that includes the title of the file from which it was extracted. You could achieve this effect with the following setting:

```
[ExtrTitle]
ProcedureStart=<$$_currtitle>: Procedure
```

If the original file's <title> string was "Setting up a customer account", the extract file's <title> section would be:

```
<title>Setting up a customer account: Procedure</title>
```

Note: If you assign [HTMLParaStyles] property Title to a paragraph format in the extract, the content of that paragraph overrides anything you specify in [ExtrTitle].

18.7.3 Replacing extracts with links in the parent file

In this section:

§18.7.3.1 [Assigning replacement code](#) on page 603

§18.7.3.2 [Using thumbnails for links to illustrations in HTML](#) on page 604

§18.7.3.3 [Supplying properties for extracted graphics](#) on page 607

18.7.3.1 Assigning replacement code

In the [ExtrReplace] section you can assign HTML code, including macros, to the ExtrStart format or **ExtrStart** marker, whichever you used to designate the start of the extract. The code you assign replaces the entire extract in the parent file. For example:

```
[HTMLParaStyles]
Heading2=ExtrStart

[ExtrReplace]
; doc format = HTML code to use instead of extracted para
Heading2=<$YourMacroForTheReplacement>
```

If you need different replacements for different extracts, you could either use different starting formats, or you could use an **ExtrReplace** marker to specify replacement code for a particular extract; the marker takes precedence over anything you specify in the [ExtrReplace] section.

You can use several predefined macro variables in replacement code to reference the replaced extract file, the extract title, and the first graphic in the extract. [Table 18-8](#) lists the variables you can use in extract replacement code. Also, predefined macro <\$_extrthumb> provides a convenient way to include scaled thumbnails of graphics as replacement links; see §18.7.3.2.3 [Providing scaled thumbnails](#) on page 605.

Table 18-8 Predefined macro variables for extract replacement code

Macro variable	Definition	Reference
<\$\$_extrgraph>	File name, as modified by any value specified for [Graphics]ExtrGraphSuffix, of the first graphic in an extract; use to include a thumbnail of the graphic	18.7.3.2.2
<\$\$_extrgraphclass>	CSS class name to use in <\$_extrthumb> macro	18.7.3.2.3
<\$\$_extrgraphhigh>	Thumbnail height in pixels, for use in <\$_extrthumb> macro	18.7.3.2.3
<\$\$_extrgraphtarget>	target attribute for window used by <\$_extrthumb>	18.7.3.2.3
<\$\$_extrgraphid>	Mif2Go internal name of the first graphic in an extract; use to reference properties	18.7.3.3

Table 18-8 Predefined macro variables for extract replacement code

Macro variable	Definition	Reference
<\$\$_extrgraphwide>	Thumbnail width in pixels, for use in <\$_extrthumb> macro	18.7.3.2.3
<\$\$_extrfile>	Extract file name	18.7.3.3
<\$\$_extrtitle>	Extract title	18.7.3.3

For example, the following code uses a thumbnail graphic to link to an extract:

```
[HTMLParaStyles]
FigCaption=Contents ExtrStart Title
FigAnchor=ExtrFinish

[ExtrReplace]
FigCaption=<$_ThumbCode>

[ThumbCode]
<p class="thumbnail"><a href="<$_extrfile>">
  " /></a></p>
```

See §18.7.3.2 [Using thumbnails for links to illustrations in HTML](#) on page 604.

18.7.3.2 Using thumbnails for links to illustrations in HTML

In extract replacement code you can provide a thumbnail—a miniature version—of the first graphic in the extract, and use the thumbnail as a link to the extract.

In this section:

§18.7.3.2.1 [Choosing a thumbnail method](#) on page 604

§18.7.3.2.2 [Providing separate thumbnails](#) on page 605

§18.7.3.2.3 [Providing scaled thumbnails](#) on page 605

§18.7.3.2.4 [Including text with a thumbnail](#) on page 606

18.7.3.2.1 Choosing a thumbnail method

The best way to provide thumbnails depends in part on which of the following you are generating:

- server-based HTML
- compiled or local-based Help system.

The issue is the thumbnail graphic itself.

Server-based systems

For server-based HTML or OmniHelp, you can include an additional smaller version of each image, to replace (and provide a link to) the original image. Providing an additional image for a thumbnail makes sense if users are downloading a page at a time; if they do not want to view the full-size image, they can avoid the time cost of downloading it.

Local-based systems

However, for a compiled Help system (such as HTML Help), or a local-based system (such as JavaHelp, Oracle Help for Java, or local HTML or OmniHelp), the additional-image method increases the file size (and download time), because of the added size of the thumbnails themselves. For these cases it can make more sense to use the original image file, but specify a smaller size, and let the browser do the scaling. The resulting thumbnail might not be as pretty but it should still be identifiable, and that is really all that is required of thumbnails.

Which method is better for a given project? It depends. The graphics count and sizes are among the factors to consider.

18.7.3.2.2 Providing separate thumbnails

If you provide separately created thumbnails, use the following option, which is the default:

```
[Graphics]
; ExtrGraphThumbnail = Named (default,
; use original name plus ExtrGraphSuffix)
ExtrGraphThumbnail=Named
```

When `ExtrGraphThumbnail=Named`, you provide a thumbnail version of the first image in each extract. You must create the thumbnails yourself, using a third-party graphics tool, and store them in the same directory with the output graphics. Each thumbnail must have the same file name as the corresponding output graphic, but with a suffix added to the base name. You specify the suffix as follows:

```
[Graphics]
; ExtrGraphSuffix = suffix for file name of first graphic in an
; extract, used in the predefined <$$_extrgraph> macro to identify
; its thumbnail
ExtrGraphSuffix=tn
```

*Name thumbnail
after output
graphic*

Except for the suffix, each thumbnail must have the same name as the corresponding output graphic. This means that if your project involves having **Mif2Go** produce graphics with generated names, you must make each thumbnail name match the **Mif2Go**-generated name (not the original name).

*Place thumbnail
in project
directory*

Each thumbnail you create must be placed in the same directory with the corresponding output graphic; this might be different from the directory where your original graphics are located.

For example, suppose your document references the following graphic:

```
D:\MyDoc\graphics\jaguar.bmp
```

And suppose **Mif2Go** generates from `jaguar.bmp` the following output graphic:

```
D:\MyDoc\HTMout\aa0f3de8.jpg
```

You must provide the following thumbnail for `jaguar.jpg`:

```
D:\MyDoc\HTMout\aa0f3de8tn.jpg
```

*Thumbnail not
found*

If **Mif2Go** does not find a properly named thumbnail for the first graphic in an extract, you get either a broken link or just the `alt` text in the replacement code.

*Graphic not
present*

If there is no graphic in the extract, the value of `<$$_extrgraph>` for that extract is `NULL`, and the literal name `extrgraph` appears in the replacement code wherever you specified `<$$_extrgraph>`.

18.7.3.2.3 Providing scaled thumbnails

When you use scaled thumbnails, the name of each thumbnail is the same as the name of the full-size graphic. To provide thumbnails scaled by the browser at run time from your original graphics, specify the following option:

```
[Graphics]
ExtrGraphThumbnail=Scaled
```

When `ExtrGraphThumbnail=Scaled`, **Mif2Go** uses the original image, applying scaling factors that you can specify:

```
[Graphics]
; ExtrGraphHigh = size in pixels for height of thumbnail
; display of graphic when ExtrGraphThumbnail=Scaled
; default 96 pixels (one inch)
ExtrGraphHigh=96
```

```

; ExtrGraphWide = size in pixels for width of thumbnail
; display of graphic when ExtrGraphThumbnail=Scaled
; default 96 pixels (one inch)
ExtrGraphWide=96
; ExtrGraphClass = name of CSS class to use in predefined
; <$_extrthumb> macro
;ExtrGraphClass=thumbnail
; ExtrGraphTarget = target attribute for window used by <$_extrthumb>
ExtrGraphTarget=_blank

```

For the thumbnail, ExtrGraph* settings override any [Graph*] settings for width and height values. The ExtrGraph* settings do not conflict with (for example) a user-defined <\$_ExtrGraphHigh> macro, nor with predefined macro variable <\$\$_extrgraphhigh> or <\$\$_extrgraphwide>; all are in different **Mif2Go** internal namespaces.

Preserve aspect ratio

If you want to use a reduced size for thumbnails, but not all images have the same aspect ratio, set only one of ExtrGraphHigh or ExtrGraphWide to the number of pixels you want, and set the other to 0 (zero).

Preserve image size

If you want the thumbnail to be the size of the original image as it appears in FrameMaker, instead of specifying width and height values, set the following option:

```

[Graphics]
; OrigSizedThumbnail = No (default)
; or Yes (use original Frame size for it)
OrigSizedThumbnail=Yes

```

When OrigSizedThumbnail=Yes, the size specified for the image in FrameMaker is used for the thumbnail instead of any size values specified with ExtrGraphHigh or ExtrGraphWide. This can be a reasonable way to present screenshots when you do not want the thumbnail to be any smaller, but you want users to have a way to make the screenshot legible. For a simpler way to accomplish the same objective, without using extracts, see §23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710.

Predefined macro <\$_extrthumb>

For convenience you can use built-in macro <\$_extrthumb>, which is defined as follows:

```

<p class="<$$_extrgraphclass>"><a href="<$$_extrfile>"
target="<$$_extrgraphtarget>"> 0)> height="<$$_extrgraphhigh>"<$_endif>\
<$_if ($$_extrgraphwide > 0)> width="<$$_extrgraphwide>"<$_endif>\
alt="<$$_extrtitle>" /></a></p>

```

Using this macro, the settings you need for scaled thumbnails can be reduced to the following:

```

[Graphics]
ExtrGraphThumbnail=Scaled

[ExtrReplace]
*=<$_extrthumb>

```

18.7.3.2.4 Including text with a thumbnail

Suppose you want to display, next to each thumbnail, text to indicate that a full-size version of the graphic is but a click away; for example, **Click to enlarge**.

If you use extraction to create the thumbnails (see §18.7.3.2.3 [Providing scaled thumbnails](#) on page 605), you can do something like the following to put the text next to the image:

```

[ExtrReplace]
Thumbfmt=<$thumbnail>

[thumbnail]
<table border="0"><tr>
  <td><$_extrthumb></td>

```

```
<td><p class="thumbtext">Click to enlarge</p></td>
</tr></table>
```

Add an entry for `p.thumbtext` to your CSS, perhaps in `[CSSStartMacro]` if you have **Mif2Go** generate CSS each time. You could also give the table a class, and define its properties in the CSS file.

18.7.3.3 Supplying properties for extracted graphics

You can use predefined macro variable `<$$_extrgraphid>` to access properties you have assigned to individual graphics in the configuration file.

For example, suppose you are using JavaScript in extract replacement code to specify characteristics of the secondary window in which each extracted graphic will appear. And suppose you want each window to be the same size as the graphic. You could place code like the following in an **ExtrReplace** marker for each individual extract, with the dimensions for that particular graphic:

```
<p class="fig"><a href="javascript:location='<$$_currfile>';
window.open('<$$_extrfile>', 'height=387,width=550')">
<$$_extrtitle></a></p>
```

Or, you could specify the dimensions of any extractable graphics in the project configuration file: (see §23.9.2 [Adjusting image size for selected graphics](#) on page 720):

```
[GraphWide]
; Graphic file name = number of pixels wide, 0 to omit width attribute
aa123456=525
ab654321=440

[GraphHigh]
; Graphic file name = number of pixels high, 0 to omit height
; attribute
aa123456=150
ab654321=220
```

Then you could access the dimensions with a list variable (see §28.4 [Using multiple-value list variables](#) on page 806) in the replacement code. For example, you could replace the JavaScript height and width clause with the following code, where `$$graphhigh` and `$$graphwide` are list variables:

```
'height=<$$graphhigh[$$_extrgraphid]>,
width=<$$graphwide[$$_extrgraphid]>'
```

You could define macros to supply default values for graphics not listed in the configuration sections that your list variables access. For example:

```
[ExtrGraphHigh]
<$$ht = ($$graphhigh[$$_extrgraphid])>
<$_if ($$ht==0)><$$ht=387><$_endif><$$ht>
```

18.7.4 Specifying extracts: an example

This example delimits figures, sidebars, and procedures to be extracted from the parent file, and specifies macros to be inserted in the extracts.

Assign extract properties and macros to paragraph formats:

```
[HTMLParaStyles]
; Extract figures and sidebars:
ArtAnchor=ExtrStart LEnd
SideBarAnchor=ExtrStart LEnd

; Extract procedures:
ProcHead=ExtrStart Title CodeAfter CodeBefore LEnd
```

```

; Put titles on the figure and sidebar extracted pages:
SideBarHeading=Title CodeBefore
ArtCaption=Title CodeBefore

; End extracts:
Body=ExtrEnd LEnd
Heading3=ExtrEnd LEnd

```

Call macros to put a Close Window button before the topic:

```

[ParaStyleCodeBefore]
SideBarHeading=<$CWbutton>
ArtCaption=<$CWbutton>

```

In the parent file, substitute links to the extracts for the extracted material:

```

[ExtrReplace]
; Replace extracted sidebars with "See..." links:
SideBarAnchor=<a href="<$$_extrfile>">See...</a>
; Replace extracted figures with "Figure..." links:
ArtAnchor=<a href="<$$_extrfile>">Figure...</a>
; Replace extracted procedures with links to the procedures:
ProcHead= <a href="<$$_extrfile>"><$$_extrtitle></a>

```

Assign macros to specify links from and within the extract files:

```

[ExtrTop]
; Place a link at the top to the More Topics section at the bottom:
ProcHead=<$MoreTopicsJump>
SideBarAnchor=<$MoreTopicsJump>
ArtAnchor=<$MoreTopicsJump>

[ExtrBottom]
; At the bottom add links back to the parent doc and to other topics:
ProcHead=<br><br><$NavListExtract><$BackToTop>
SideBarAnchor=<br><br><$NavListExtract><$BackToTop>
ArtAnchor=<br><br><$NavListExtract><$BackToTop>

[ExtrHead]
; Add a link to the style sheet that overrides selected formats:
ProcHead=<link rel="stylesheet" href="Ovr.css" type="text/css">
SideBarAnchor=<link rel="stylesheet" href="Ovr.css" type="text/css">
ArtAnchor=<link rel="stylesheet" href="Ovr.css" type="text/css">

```

19 Creating HTML links

This section shows how to provide basic links in HTML output. Topics include:

- §19.1 [Understanding sources of links](#) on page 609
- §19.2 [Specifying link appearance](#) on page 609
- §19.3 [Specifying link destination](#) on page 613
- §19.4 [Creating jumps to particular windows for HTML](#) on page 616
- §19.5 [Converting FrameMaker links to HTML](#) on page 617
- §19.6 [Linking to other files and other Mif2Go projects](#) on page 621
- §19.7 [Linking to external destinations](#) on page 625

See also:

- §20 [Providing navigation in HTML](#) on page 627

19.1 Understanding sources of links

Mif2Go creates HTML links from the following items in FrameMaker:

<i>Cross references:</i>	Cross references are converted to links to cross-reference sources; see §19.5.1 Converting FrameMaker cross references to HTML on page 617.
<i>Hypertext internal links:</i>	Hypertext links (gotolink and openlink link to newlink) are activated in HTML; see §19.5.2 Converting FrameMaker hypertext links to HTML on page 619.
<i>ObjectID links:</i>	Links created by FrameMaker for TOC, Index, and other generated lists are enabled; see §19.5.3 Including ObjectID anchors as link targets on page 620.
<i>Hypertext “message” commands:</i>	Hypertext message URL and message openlink are activated in HTML; see §19.7 Linking to external destinations on page 625.
<i>Paragraph formats:</i>	Links connecting a hierarchy of headings can form a “breadcrumb trail”; see §20.2 Generating trails of links on page 627.

19.2 Specifying link appearance

Link presentation is typically set in the browser, by the user. If you do nothing, links come out the color the user specifies; with or without underlines, as the user chooses. It is best *not* to impose your own ideas on users in this area.

In this section:

- §19.2.1 [Specifying link colors](#) on page 610
- §19.2.2 [Specifying link class](#) on page 610
- §19.2.3 [Assigning link attributes with markers](#) on page 612
- §19.2.4 [Specifying link properties with macros](#) on page 612
- §19.2.5 [Replacing problem characters in links](#) on page 612
- §19.2.6 [Forcing link text to lowercase](#) on page 613

19.2.1 Specifying link colors

If you really must specify link color for some reason, always use the attributes intended for this purpose, either in the <body> tag or in CSS (see §22 [Setting up CSS for HTML](#) on page 681). You must specify three color values: for unvisited links, for active (already selected) links, and for visited links. For example:

```
<body link="#0000ff" alink="#ff0000" vlink="#800080" >
```

The defaults of blue for `link` and purple for `vlink` (the default for `alink` varies) are best left alone unless you have a compelling reason to use something else. An `alink` is in an “active” state only while the mouse is clicked on it with the button held down. The rest of the time, it is either unvisited or visited.

Set via <body>
tag

You can set link appearance in the attributes for the <body> tag. For example:

```
[Attributes]
; link = hyperlink active color,
; alink = hyperlink color when clicked,
; vlink = visited hyperlink color, and
; #..... = your hex color for any of these
body= bgcolor="#FFFFFFE1" text="#000080" link="#008020" vlink="#804000"
```

Keep all attributes for a given element on one line, regardless of line length.

Set via CSS

In CSS (browser-dependent at present) you could use, for example:

```
a:link { color: blue }
a:active { color: red }
a:visited { color: #800080 }
```

To override CSS-defined colors for some or all links, see §19.2.2 [Specifying link class](#) on page 610.

19.2.2 Specifying link class

To give some of the links in your document an appearance different from that produced by the “a:” class properties specified in CSS or the default browser settings, you can name and define other CSS classes, and apply them selectively to the links in your document.

For example, suppose you want certain links to be red except when the mouse hovers over them, when they should change to green underlined. In CSS you might define link class `traffic`:

```
a.traffic:link,a.traffic:visited,a.traffic:active {color: #ff0000;}
a.traffic:hover {color: #00ff00; text-decoration: underlined;}
```

You can apply such a class to selected links via marker in FrameMaker, or via paragraph-format assignment in the **Mif2Go** configuration file. To change just one or two links, probably a marker is easier. To change many links, you might want to use a special paragraph format for the text where the links occur.

In this section:

§19.2.2.1 [Assigning a link class with a marker](#) on page 610

§19.2.2.2 [Assigning a link class with a paragraph format](#) on page 611

19.2.2.1 Assigning a link class with a marker

You can use custom FrameMaker marker **LinkClass** to assign a CSS class to a single link, as follows:

1. Create a FrameMaker marker type named **LinkClass** (see §29.2 [Adding custom marker types](#) on page 832).

2. For each link to be altered:
 - 2.1. Place a **LinkClass** marker in your document, just before the link.
 - 2.2. Make the content of the marker the name of the CSS class you want applied.

For example, to apply CSS class `traffic` to a particular link, somewhere before the link you would insert a **LinkClass** marker with content:

```
traffic
```

CSS class `traffic` would be applied (only) to the next link after the marker:

```
<a class="traffic" ...>link text</a>
```

See also:

§19.2.3 [Assigning link attributes with markers](#) on page 612

§25.3.3 [Assigning WAI link attribute values with custom markers](#) on page 759

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

19.2.2.2 Assigning a link class with a paragraph format

You can cause all links in your document to inherit the class properties of the paragraphs where the links occur, or you can assign a CSS class to all the links that occur in paragraphs of a particular format.

To make all links use the same CSS class properties as their containing paragraphs:

```
[CSS]
; LinkClassIsParaClass = No (default)
; or Yes (adds the same class attribute as is used for
; the current para to all links within that para)
; Default is reversed to Yes if UseCSS=Yes.
LinkClassIsParaClass=Yes
```

When you use CSS, the default value of `LinkClassIsParaClass` is reversed to `Yes`; see §22.5 [Understanding how CSS affects other options](#) on page 687.

To assign a class to the links in a particular paragraph format:

```
[HTMLParaStyles]
; ParaLinkClass uses the name in [StyleParaLinkClass]
; as the class attribute of the contained links; if none is
; specified, it uses the same class as the para itself
ParaFmt=ParaLinkClass

[StyleParaLinkClass]
; doc style = name to use in the class attribute
; of the links in the para
ParaFmt=classname
```

A `ParaLinkClass` assignment overrides any `LinkClassIsParaClass` setting in `[HTMLOptions]`.

For example, to assign CSS class `traffic` to all links that occur in text with paragraph format *Blurb*, and cause all links in *Intro* paragraphs to look just like the rest of *Intro* text:

```
[HTMLParaStyles]
Intro=ParaLinkClass
Blurb=ParaLinkClass

[StyleParaLinkClass]
Blurb=traffic
```

A *Blurb* paragraph that contains a link would convert to HTML like this:

```
<p class="blurb">Text containing a link to <a class="traffic" href=
"#">somewhere</a>.</p>
```

An *Intro* paragraph that contains a link would convert like this:

```
<p class="intro">Text containing a link to <a class="intro" href=
"#">somewhere</a>.</p>
```

19.2.3 Assigning link attributes with markers

To assign an HTML attribute to a link, just before the link insert a marker with a name that starts with **Link** and ends with the name of the link attribute. Make the content of the marker the value of the named attribute. **Mif2Go** puts the attribute and its value in the generated `<a>` tag.

See also:

§19.2.2.1 [Assigning a link class with a marker](#) on page 610

§25.3.3 [Assigning WAI link attribute values with custom markers](#) on page 759

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

19.2.4 Specifying link properties with macros

To include a macro in the `href` attribute of HTML links, assign the `LinkSrc` property to any paragraph formats that contain links you wish to modify:

```
[HTMLParaStyles]
ParaFmt = LinkSrc
```

To specify the macro code:

```
[ParaStyleLinkSrc]
ParaFmt = code for the href attribute value
```

You can include macro variable `<$$_linksrc>` in the macro to insert the default content of the `href` attribute.

For example, suppose you want to use JavaScript for all links that occur in *Body* paragraphs, changing the links from the default format:

```
<a href="destination">Some text</a>
```

to this format:

```
<a href="javascript:LinkTo('destination');">Some text</a>
```

To make the links look like this in HTML output:

```
[HTMLParaStyles]
Body = LinkSrc

[ParaStyleLinkSrc]
Body = javascript:LinkTo('<$$_linksrc>');
```

Macro variable `<$$_linksrc>` provides the original *destination* value for the `href` attribute. **Mif2Go** supplies the double quotes around the entire attribute value; do not include them in the macro definition. Any quote marks needed *within* the macro must be single quotes.

Macros are also helpful when you need more than one line of `href` attribute information, or when you want to use the same `href` attributes in many different configuration files.

See §28 [Working with macros](#) on page 787.

19.2.5 Replacing problem characters in links

Some characters that are acceptable in FrameMaker hypertext links and cross references cause problems for browsers; for example, HTML insists on all-lowercase IDs. **Mif2Go** processes FrameMaker hypertext link and cross-reference markers to ensure acceptable

IDs, similar to the way CSS class names are processed; see §22.7.1 [Understanding CSS class name restrictions](#) on page 691.

Spaces are removed or replaced

Part of the job is to remove all spaces, possibly replacing them with another character when that is necessary to prevent name clashes. You can specify any alphanumeric character (or a hyphen or an underscore) to replace spaces.

To set the character used to replace spaces in links:

```
[HTMLOptions]
; These alphanumeric chars are used as space replacements in IDs;
; if non-alphanumeric (other than hyphen or underscore), spaces are
; stripped instead (default)
; XrefSpaceChar = alphanumeric char to use in xref markers
XrefSpaceChar=-
; HyperSpaceChar = alphanumeric char to use in hyperlinks (not URLs)
HyperSpaceChar=-
```

By default, **Mif2Go** removes spaces without replacing them. The same thing happens if you set `XrefSpaceChar` or `HyperSpaceChar` to any non-alphanumeric character other than a hyphen or an underscore: **Mif2Go** removes all spaces without replacing them.

19.2.6 Forcing link text to lowercase

To make sure all hypertext links are lowercase in HTML output:

```
[HTMLOptions]
; MakeFileHrefsLower = No (leave case unchanged) or Yes
MakeFileHrefsLower = Yes
```

`MakeFileHrefsLower` is set to `Yes` in system configuration file `d2htm_config.ini`. If you want **Mif2Go** to leave case alone in hypertext links, you must override this setting in a project or local configuration file.

`MakeFileHrefsLower` applies to hypertext links and interfile cross references. When you use the `FileName` property to name the section that contains the cross-reference destination, the setting applies also to cross references within the same file (see §34.8.4.4 [Creating special paragraph formats to name output files](#) on page 950).

Case mismatch can cause links to fail

If you change the case of a FrameMaker file name in your document after you have created cross references or hypertext links to that file, the original case is preserved in the markers. If your HTML output will be deployed on a UNIX system, links based on those markers will fail because of the case mismatch. For JavaHelp, Oracle Help for Java, and Eclipse Help (which is based on Java), the links fail on Windows systems, also. Java was created by a UNIX company (Sun Microsystems) and uses UNIX rules, in which file-name case is always significant.

19.3 Specifying link destination

In this section:

- §19.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 614
- §19.3.2 [Forcing all links to top-of-page](#) on page 614
- §19.3.3 [Linking to an arbitrary location](#) on page 614
- §19.3.4 [Providing alternate link destinations](#) on page 615
- §19.3.5 [Troubleshooting bad links](#) on page 616

See also:

- §19.6 [Linking to other files and other Mif2Go projects](#) on page 621

19.3.1 Forcing links to top-of-page for selected paragraph formats

You can specify that all interfile links in a particular paragraph format should go to the start of the target page instead of to the cross-reference marker or hypertext **newlink** marker location on the page, by assigning property NoRef to the format. For example:

```
[HTMLParaStyles]
zNextSection=NoRef
zPrevSection=NoRef
```

You can assign property NoRef to cross-reference formats, also; see §19.5.1.3 [Specifying HTML options for selected cross-reference formats](#) on page 618.

19.3.2 Forcing all links to top-of-page

You can specify that all interfile links should go to the start of the target page rather than to the cross-reference marker or hypertext **newlink** marker location. This is equivalent to setting NoRef for all paragraph styles in [HTMLParaStyles]; see §19.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 614:

```
[HTMLOptions]
; NoLocations = No (default)
; or Yes (suppresses the part of all links after the filename)
NoLocations = Yes
```

To remove all named anchors also:

```
[HTMLOptions]
; RemoveANames = No (default)
; or Yes (DITA, eliminate <a name=...> tags)
RemoveANames = Yes
```

19.3.3 Linking to an arbitrary location

To create a link to an arbitrary location in a file, you must identify or establish a target at that location. You can use an existing FrameMaker ObjectID as the target, and in the reference follow the file name with # then the ObjectID:

```
<a href="filename#ObjectID"></a>
```

However, a FrameMaker ObjectID will change if you do any of the following to the paragraph containing the object (see §5.3.2 [Working with FrameMaker ObjectIDs](#) on page 118):

- copy and paste the paragraph (the pasted copy gets a new ID)
- delete the paragraph and then retype it
- hide the paragraph with a condition and then show it.

You can also use a format macro to create and name the target. For example, suppose one of your HTML files includes a procedure, and you want to create a link to the procedure rather than to the beginning of the file. Suppose your procedures always start with a paragraph format called *ProcHead*. You could assign the following properties and code to *ProcHead*:

```
[HTMLParaStyles]
ProcHead=CodeBefore

[ParaStyleCodeBefore]
ProcHead=<a name="startproc" id="startproc"></a>
```

If the procedure is in HTML file xx123456, the link would look like this:

```
<a href="xx123456#startproc"></a>
```

You must ensure the target file contains no conflicting uses of the same target name, for example in **newlinks**.

19.3.4 Providing alternate link destinations

You can use a macro to place a copy of a paragraph ID or a cross-reference ID somewhere in the generated HTML code other than immediately before the content of the paragraph in question. Two predefined macro variables capture the values of these **Mif2Go**-generated IDs:

`<$$_parauid>` Paragraph ID of the current paragraph
`<$$_xrefid>` Cross-reference ID (if any) of the current paragraph

Each returns what would normally go in the `` tag:

`Xaannnnnn` for a paragraph ID
`Raannnnn` for the ID of the first cross-reference marker in a paragraph

where:

`aa` is the FileID of the HTML file (provided
`[HTMLOptions]UseFileIDs=Yes`, which is the default)
`nnnnnnn` is the ObjectID of the paragraph or cross reference.

For example, if the anchor **Mif2Go** creates from the first cross-reference marker in a paragraph looks like this:

```
<a name="Rxxaa39983">
```

The value of `<$$_xrefid>` for that paragraph would be:

```
Rxxaa39983
```

Note: FrameMaker ObjectIDs can change; see §19.3.3 [Linking to an arbitrary location](#) on page 614.

See §5.3 [Identifying files and objects](#) on page 117.

*Move a jump
destination*

Suppose you want to display something (perhaps a table, a link, or a graphic) above a mid-topic (non-split) heading, so that jumps to that heading show the preceding something at the top of the window, above the heading.

You could use `[ParaStyleCodeBefore]` to place HTML code just before the content of each such mid-topic heading, and include a copy of the paragraph ID (or cross-reference marker ID) in that code. This would force hypertext jumps (or cross references) to the paragraph to go to the preceding code instead of to the paragraph content. Whatever is produced by that code would then appear at the top of the window when you click a link to the paragraph.

For example, to precede a mid-topic heading with a one-cell table containing a link back to a local TOC:

```
[HTMLParaStyles]
SubHead=CodeBefore

[ParaStyleCodeBefore]
SubHead=<p><a name="<$$_xrefid>"></a></p><$BackToTOC>

[BackToTOC]
<table cellpadding="5" border="1" cellspacing="2">
  <tr>
    <td><font size=-3><a href="#contents">Contents</a></font></td>
```

```
</tr>
</table>
```

Clicking a cross-reference link to any *SubHead* paragraph would show a box containing a link to **Contents** above the *SubHead* paragraph.

See also:

§28.3.4 [Using predefined macro variables](#) on page 800

§28.9.3 [Surrounding or replacing text with code or macros](#) on page 822

19.3.5 Troubleshooting bad links

If you find that a link always jumps to the top of a file even when the destination is specified as mid-file, check for the presence of non-alphanumeric characters in the link; see §19.2.5 [Replacing problem characters in links](#) on page 612.

However, usually the reason a link takes you to the top rather than the desired location is that the anchor part, after the #, is wrong; see §19.3.3 [Linking to an arbitrary location](#) on page 614.

Also check that the destination anchor is actually present in the destination file; see §19.6 [Linking to other files and other Mif2Go projects](#) on page 621, and §19.7 [Linking to external destinations](#) on page 625.

See also:

§5.1.5 [Checking for broken links in HTML or XML output](#) on page 112

19.4 Creating jumps to particular windows for HTML

You can assign a particular window type as a jump destination. A window assignment can specify jumps the following ways:

- all jumps from a character or paragraph format
- all jumps to a particular file or URL
- individual jumps to a particular window.

A window assignment supplies a value for the `target` attribute of the `` tag **Mif2Go** generates for the jump. If you are using framesets that value must be the name of a frame (see §13.14 [Using framesets](#) on page 450), possibly one of several names reserved by JavaScript, such as `_top` or `_blank`:

- A jump to `_top` gets you out of a frameset; it does not open a new window.
- A jump to `_blank` always opens a new window.

A jump to a window with a non-reserved name, if the window is not in the current frameset (if any), opens a window of that name; and the next jump to the same name reuses that same window. You can specify target windows the following ways:

[Specify window by jump format](#)

[Specify window by jump destination](#)

[Specify window with a marker.](#)

See also:

§7.7 [Jumping to secondary windows in Help systems](#) on page 224

§7.8 [Creating pop-up topics for Help systems](#) on page 225

*Specify window
by jump format*

You can use a character format to mark all jumps to a particular window type. For example:

```
[Targets]
; doc format = name of frame to use for jumps from within this style
; For OmniHelp ALink and KLink jumps, targets make no sense
; and are ignored.
JumpNew=_blank
```

If you know that such jumps always occur in a particular type of paragraph, such as *Step* paragraphs in procedures, you could use a paragraph format. For example:

```
[Targets]
Step*=procwin
```

*Specify window
by jump
destination*

If you know that all jumps to a particular HTML page (such as `glossary.htm`) should go to a particular window type, you can specify the window to use for that page. For example:

```
[TargetFiles]
; filename (no ext) or URL destination = target frame to be used
; a URL destination is the last element in the URL (no extension)
glossary=glosswin
```

*Specify window
with a marker*

If you need case-by-case handling of jumps to other windows, put a marker of type **LinkTarget** (see §29.2 [Adding custom marker types](#) on page 832), with marker content the name of the window, anywhere before the relevant hypertext jump marker.

19.5 Converting FrameMaker links to HTML

Mif2Go automatically converts FrameMaker cross references and hypertext links to `` links in HTML output. You can specify options for these links.

In this section:

§19.5.1 [Converting FrameMaker cross references to HTML](#) on page 617

§19.5.2 [Converting FrameMaker hypertext links to HTML](#) on page 619

§19.5.3 [Including ObjectID anchors as link targets](#) on page 620

19.5.1 Converting FrameMaker cross references to HTML

You can specify settings to prevent some cross references from being converted to links, to customize cross-reference behavior, or to direct **Mif2Go** to produce diagnostic information about broken cross-reference links.

In this section:

§19.5.1.1 [Identifying cross-reference markers](#) on page 617

§19.5.1.2 [Specifying HTML options for all cross references](#) on page 618

§19.5.1.3 [Specifying HTML options for selected cross-reference formats](#) on page 618

See also:

§19.6 [Linking to other files and other Mif2Go projects](#) on page 621

19.5.1.1 Identifying cross-reference markers

FrameMaker identifies cross-reference markers with a unique reference number; **Mif2Go** prefixes the letter “R” and the FileID to this number, and discards the rest of the marker text. If you create markers without reference numbers, **Mif2Go** prefixes your marker text with “R” and the FileID, and uses the full marker text, up to a 127-character limit. **Mif2Go** truncates marker text at 127 characters.

See also:

§5.3.4 [Working with Mif2Go FileIDs](#) on page 119

§19.6.1 [Identifying HTML link destinations with FileIDs](#) on page 621

19.5.1.2 Specifying HTML options for all cross references

You can specify several processing options that apply to all cross references:

```
[HTMLOptions]
; XrefType = Full (default) or Numeric (shorter, saves space)
XrefType=Numeric
; RemoveFilePaths = Yes (default, strip hyperlink and xref paths)
; or No
RemoveFilePaths=No
; ListMissingRefs = No (default)
; or Yes (identify missing xrefs to stderr)
ListMissingRefs=No
; CheckAllRefs = Yes (default, even if they seem unchanged)
; or No
CheckAllRefs=Yes
```

See also:

§19.6.2 [Retaining file paths in interfile links](#) on page 622

19.5.1.3 Specifying HTML options for selected cross-reference formats

You might not want every cross reference in your document to become a link in the HTML output, or you might want to specify HTML attributes for the links generated from some cross references. You can choose to have **Mif2Go** delete cross references of a certain format, convert them to text, redirect them to top-of-page, or enhance them with link attributes:

```
[XrefStyles]
; xref format name = properties (Delete, Text, NoRef, or LinkSrc)
; if omitted, xref is treated as link
```

These properties have the following effects:

- | | |
|---------|---|
| Delete | Omits the cross reference entirely from HTML output. You can assign this property to remove unwanted page references, provided you have set up the format so that deleting the cross-reference content leaves readable text. |
| Text | Prevents creation of the tag, so the cross reference does not become a link. You might want to assign this property to a cross-reference format you have used to insert short insets in FrameMaker. |
| NoRef | Creates the tag, but omits any hash part of the href attribute; for example, <i>file.htm#heading</i> would become just <i>file.htm</i> . The result is that the jump goes to the start of the file, not to a point within the file. Assign this property if you want the top of the page to show, instead of the referenced object, when a jump goes to a split file. |
| LinkSrc | Allows a Mif2Go macro in the href attribute of the link; you define the macro in section [XrefStyleLinkSrc]. If you assign property LinkSrc to a cross reference, and also to a character format applied to the same cross reference (see §19.2.4 Specifying link properties with macros on page 612), the cross-reference LinkSrc macro prevails. In the macro definition you can use predefined macro variable <\$\$_linksrc>, which provides the normal href content of the link. |

For example:

```
[XrefStyles]
zSectionLink=NoRef
Heading & Page=Text
Page=Delete
```

With these settings, **Mif2Go** would do the following:

- Omit the `#heading` part of the `href` attribute from any link generated from a cross reference that uses the `zSectionLink` format.
- Render any cross reference that uses the *Heading & Page* format as plain text rather than as a link.
- Omit any cross reference that uses the *Page* format.

To use macros (see §28 [Working with macros](#) on page 787) in the `href` attribute of the links generated from cross references, you assign property `LinkSrc` to the cross-reference format, and you specify the macro in the following section:

```
[XrefStyleLinkSrc]
; xref format name = text macro to use in the href attribute
; of the xref link; <$$_linksrc> is available to use in the macro,
; with the normal href content.
```

To assign properties `LinkSrc` and `NoRef` to paragraph formats, see:

§19.2.4 [Specifying link properties with macros](#) on page 612

§19.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 614.

19.5.2 Converting FrameMaker hypertext links to HTML

Mif2Go imposes an internal limit of 255 characters on HTML anchor names (the part after the “#”).

In this section:

§19.5.2.1 [Understanding how Mif2Go treats hypertext links](#) on page 619

§19.5.2.2 [Understanding when to use `openlink` or `gotolink`](#) on page 619

See also:

§5.10 [Creating hotspots for hypertext links](#) on page 138

§34.1.2 [Using markers to add links and instructions](#) on page 935

19.5.2.1 Understanding how Mif2Go treats hypertext links

Mif2Go converts most FrameMaker hypertext links into HTML links. Many kinds of hypertext markers are available:

- Hypertext jumps use **gotolink**, **openlink**, or one of their cousins (such as **gotolinkfitwin**) to link to a matching **newlink** marker. The **:firstpage** and **:lastpage** forms are converted into simple links to the beginning of the file named.
- Message URL markers are passed through untouched as normal `` links. Use these for links to external Web sites, **mailto:** links, and so forth.
- Markers of the **gotopage** type cause a jump to the start of what was the first paragraph on the specified page, even if the file has been split.
- Markers for **previouspage**, **nextpage**, **previouslink**, **opennew**, **popup** (menu), **matrix**, **quit**, **quital**, and **exit** are always ignored.
- Alert boxes are used only for MS HTML Help, where they provide simple pop-ups. They are not effective in generic HTML.

19.5.2.2 Understanding when to use `openlink` or `gotolink`

The syntax for **openlink** is ambiguous. The FrameMaker *Hypertext* dialog says it is:

```
openlink filename:linkname
```

which could mean either:

```
openlink [filename:]linkname
```

(link name required, file name optional) which is how **gotolink** works, or:

```
openlink filename[:linkname]
```

(file name required, link name optional) which how **openlink** works.

Use **gotolink** for targets within the same FrameMaker file.

Use **openlink** only for targets that are in a different FrameMaker file; always specify the file name, including the .fm extension. If your document uses **openlink** hyperjumps specified with a single argument that is the name of a file, and not the name of a link, set this option:

```
[HtmlOptions]
; OpenlinkIsFile = No (default) or Yes (if no colon, dest is filename)
OpenlinkIsFile=Yes
```

See §34.1.2 [Using markers to add links and instructions](#) on page 935.

19.5.3 Including ObjectID anchors as link targets

When FrameMaker generates a TOC, IX, or other list or index, and you check the box to request hypertext links, FrameMaker inserts jumps to the referenced items, using their ObjectIDs as targets. By default, **Mif2Go** includes in HTML output an anchor for every FrameMaker ObjectID, for references from the TOC and IX; and also includes the ObjectIDs of all tables and anchored frames. See §5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117.

To specify whether **Mif2Go** should include in HTML output all, a subset, or none of the ObjectIDs from your FrameMaker document:

```
[HTMLOptions]
; ObjectIDs = All (default), Referenced, or None
```

Values for ObjectIDs have the following effects:

<code>ObjectIDs=All</code>	An anchor is created for every ObjectID in your document.
<code>ObjectIDs=Referenced</code>	Anchor are created only for ObjectIDs that appear to be referenced in your document.
<code>ObjectIDs=None</code>	Anchor for ObjectIDs are not included in HTML output.

When is a default value not the default?

`ObjectIDs=All` is the default value, provided your configuration file has *no setting at all* for ObjectIDs. However, when generating a new configuration file at set-up time, **Mif2Go** specifies `ObjectIDs=All` only for HTML-based Help formats, and specifies `ObjectIDs=Referenced` for all other formats.

See §4.3 [Understanding where project settings come from](#) on page 102.

ObjectIDs=All

Keeping all ObjectIDs in the output can result in significant HTML bloat from all the `` tags. However, you need this setting if you are converting a FrameMaker-generated index to HTML-based Help and you use the following option:

```
[Index]
KeywordRefs=Para
```

See §7.5.8 [Specifying index link destinations for HTML-based Help](#) on page 215.

<i>ObjectIDs=Referenced</i>	<p>Although <code>ObjectIDs=Referenced</code> reduces anchor bloat in the output, there are other considerations if you are converting either of the following to HTML-based Help:</p> <p>FrameMaker-generated TOC</p> <p>FrameMaker-generated IX</p>
<i>FrameMaker-generated TOC</i>	<p>When you specify <code>ObjectIDs=Referenced</code>, you must tell Mif2Go which paragraph formats are referenced from the TOC. Assign the <code>Contents</code> property to each TOC-referenced paragraph format, along with any other properties needed. For example:</p> <pre>[HTMLParaStyles] ; Contents (when ObjectIDs=Referenced) causes the ObjectID to be ; retained so that FM-generated links from the TOC will work heading 1=Contents Split Title heading 2=Contents</pre> <p>See §7.4.3 Including contents entries in HTML-based Help on page 209.</p>
<i>FrameMaker-generated IX</i>	<p>When you specify <code>ObjectIDs=Referenced</code>, you might lose index links if you also set <code>[Index]KeywordRefs=Para</code>. See §7.5.8 Specifying index link destinations for HTML-based Help on page 215 and §13.8.2.2 Including paragraph references on page 445.</p>
<i>ObjectIDs=None</i>	<p><code>ObjectIDs=None</code> disables several features, such as file-splitting, that require <code>ObjectIDs</code>. Therefore, <i>this property is deprecated: do not specify <code>ObjectIDs=None</code></i>. See §18 Splitting and extracting files on page 585.</p>

19.6 Linking to other files and other Mif2Go projects

With default configuration settings, **Mif2Go** successfully converts cross references and hypertext links within and between HTML files generated in the same project. However, you might need additional settings if your project includes any of the following:

- links to or from files in other projects
- links to files whose names or locations will change after conversion
- links to FrameMaker text-inset files.

In this section:

§19.6.1 [Identifying HTML link destinations with FileIDs](#) on page 621

§19.6.2 [Retaining file paths in interfile links](#) on page 622

§19.6.3 [Enabling links to renamed or relocated files](#) on page 622

§19.6.4 [Enabling links to files in other projects](#) on page 623

§19.6.5 [Updating links between files in different projects](#) on page 624

§19.6.6 [Mapping links to text insets](#) on page 624

See also:

§19.7 [Linking to external destinations](#) on page 625

§20 [Providing navigation in HTML](#) on page 627

§C.5 [Working with reference files for HTML or XML](#) on page 1027

19.6.1 Identifying HTML link destinations with FileIDs

Unless your project consists of only one file, with no cross references to other HTML files, use the following default setting:

```
[HTMLOptions]
; UseFileIDs = Yes (default, xrefs and ObjectIDs) or No (single file)
UseFileIDs=Yes
```

When `UseFileIDs=Yes`, **Mif2Go** includes a `FileID` in the link code, so links do not get confused if a cross-reference number or `ObjectID` is not unique. **Mif2Go** assigns `FileIDs` to your FrameMaker files; see §5.3.4 [Working with Mif2Go FileIDs](#) on page 119.

19.6.2 Retaining file paths in interfile links

By default, **Mif2Go** removes paths from interfile links in your FrameMaker files. If your HTML files will be maintained in a directory structure *identical* to the structure used for the FrameMaker files from which they are generated, you must direct **Mif2Go** to retain the paths in all interfile links, and then move the output files after conversion.

To retain file paths in interfile links:

```
[HTMLOptions]
; RemoveFilePaths = Yes (default, strip hyperlink and xref paths)
; or No
RemoveFilePaths = No
```

When `RemoveFilePaths=Yes` (the default), **Mif2Go** places all HTML files in the project directory, regardless of where the originating FrameMaker files are located. Links work as created, regardless of the original directory structure. A problem arises only if both of the following are true:

- Your FrameMaker files are in a non-flat directory structure (some FrameMaker files are in different directories).
- You move the resulting HTML files to an identical directory structure.

Note: You might still get link errors (see §5.1.5 [Checking for broken links in HTML or XML output](#) on page 112) for links between FrameMaker files, especially if you are using the `CodeStore` property (see §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822); however, the links should work.

When `RemoveFilePaths=No`, you must place HTML output files in a directory structure on the target system that is identical to your FrameMaker directory structure. This means that after conversion, you must move HTML files from the project directory to other directories that correspond in name and relative position to the directories where the FrameMaker files are located. You can do this with system commands; see §34.4 [Executing operating-system commands](#) on page 937.

19.6.3 Enabling links to renamed or relocated files

You might run into situations where the names of FrameMaker files are not usable for HTML files. For example, files on UNIX systems must not have spaces in their names. And for HTML Help, the use of underscores in names seems to manifest defects.

Note: To stay out of trouble, restrict file names to letters and digits only, no spaces or other characters. See §1.1.2 [File, directory, and path names](#) on page 51.

If you need to rename HTML files *after* **Mif2Go** produces them, you must tell **Mif2Go** the names (and possibly the file paths) to use for the renamed files in links. This step is essential if any links exist between the renamed files and other files in the project, or other files in another directory. For example:

```
[XrefFiles]
; original filename (no ext) = html filename (no ext, path OK)
Code1 = ../codes/federal/Code1
Cover = 00begin
```

Note: Even if you rename a file in `[XrefFiles]`, **Mif2Go** goes by the original FrameMaker name in all other sections of the configuration file.

Entries in `[XrefFiles]` replace the `href` link to the file named on the left of the equals sign (base name) with the path and name on the right. Therefore you can also use this method to provide paths to files that will be relocated to a directory different from the main project directory.

19.6.4 Enabling links to files in other projects

You can create HTML output from one **Mif2Go** project that has active links to HTML files in another **Mif2Go** project, provided the two projects meet the following requirements:

- For each project, all HTML files generated from a given FrameMaker file are in the same directory as the corresponding reference (`.ref`) file (see §C.5 [Working with reference files for HTML or XML](#) on page 1027).
- FrameMaker file names in the two projects are unique (for example, you cannot have files in both called `Intro.fm`), unless there are no inter-project links to or from files that have the same name.

FileID files must match

All FrameMaker files in *both* projects must be listed in the *same* FileID file (`mif2go.ini`); see §5.3.4.1 [Understanding how and where FileIDs are assigned](#) on page 120.

To reference the same FileID file from the configuration files for both projects, include the following setting in each project configuration file:

```
[Setup]
; IDFileName = name of file that contains FileIDs for this project
IDFileName=F:/path/to/combined/mif2go.ini
```

See §C.4 [Renaming or relocating the Mif2Go FileID file](#) on page 1027.

Reference files must reciprocate

If the two projects produce HTML files (and reference files) in different project directories, for each project you must tell **Mif2Go** where to find the appropriate reference files that go with the FrameMaker files in the other project. For example:

```
[RefFiles]
; original .fm filename = path to directory containing its .ref file
omnihelp=g:/omnisys/ug/oh
ohdesign=g:/omnisys/omhelp
```

You must include `[RefFiles]` sections in the configuration files for *both* projects, each with entries that point to the location of `.ref` files in the other project.

Wildcards in file names

You can use wildcards in the FrameMaker file names you specify in `[RefFiles]`, provided the result does not inadvertently subsume the name of any FrameMaker file in the other project. The settings in section `[RefFiles]` are the first things **Mif2Go** checks.

As a worst case, if you were to specify `*=some/other/directory`, **Mif2Go** would do the following:

- direct *all* interfile links in the second project to `some/other/directory`
- write *all* `*.ref` files for the second project to `some/other/directory`.

This is a spectacularly Bad Idea.

If a reference file is not found

If **Mif2Go** cannot find (or create) a reference file, links to any HTML files split from the FrameMaker file in question revert to links to the base name of that FrameMaker file, with extension `.htm`; and those links do not work.

To find a reference file, **Mif2Go** looks in the following places, in this order:

- the directory (if any) you specified in `[RefFiles]`
- the directory where the FrameMaker file in question resides

- the project directory for the current conversion.

If you specify a path in `[RefFiles]`, but the required reference file is not in the specified location, **Mif2Go** creates a reference file in that directory. **Mif2Go** goes on to look at other possibilities only if writing the new reference file fails, perhaps due to server permissions.

If **Mif2Go** cannot create a reference file in the directory you specified, **Mif2Go** looks in the directory where the link is pointing: at the FrameMaker file. If the reference file is not there, **Mif2Go** looks for a matching MIF file in the same directory; if a MIF file is present, **Mif2Go** creates a reference file in that directory.

If the write fails this time, or if there is no MIF file, **Mif2Go** looks in the current project directory. If the reference file is not there, **Mif2Go** creates a reference file in the current project directory.

19.6.5 Updating links between files in different projects

For each FrameMaker file, **Mif2Go** requires both the corresponding `.ref` file, and all the `.htm` files produced from that FrameMaker file, to reside in the same directory as the `.ref` file itself. Then when you convert a different book, and links from an external file change because a split name changed, **Mif2Go** updates the files of the referencing book at the same time as the referenced book. This way you do not have to reconvert the other books.

However, if you are using wrap directories (see §35.6 [Assembling files for distribution](#) on page 961), you do have to copy the changed files from their project directories to the corresponding wrap directories. **Mif2Go** does not copy the changed files to the wrap directories for the other books. To automate this process, create a `.bat` file that copies all the output files of all the books to their final destinations, and run that as a `SystemEndCommand` (see §34.4 [Executing operating-system commands](#) on page 937) whenever you rerun any book in the group. Copying the files should take only a few seconds to run.

19.6.6 Mapping links to text insets

If your document includes text insets imported by reference, and contains links to those text insets, provide settings in section `[XrefFiles]` to map the insets to their containers.

For example, suppose `Mydoc.book` contains the following files:

```
Chap1.fm, which imports insets from file Boiler.fm
Chap2.fm
Chap3.fm, which imports insets from file Trouble.fm
```

Suppose you have cross references or other links (working properly in FrameMaker) to, from, and between insets. You would specify these settings in the configuration file:

```
[XrefFiles]
Boiler=Chap1
Trouble=Chap3
```

That is, you need only specify the *container file* for each inset. As long as all cross-reference markers (and hypertext **newlinks**) in insets are accessible from the container by normal means (for example, **gotolink**), this is all you need to do. If links are made only *from* insets *to* their containers, you do not need any `[XrefFiles]` entries.

See also:

§2.5.4 [Setting up cross references to and from text insets](#) on page 70

19.7 Linking to external destinations

To include a link in your FrameMaker document to a Web site, to a PDF file, or to some other destination, use a hypertext “message” command. **Mif2Go** produces HTML links from hypertext markers in your document that contain the following types of hypertext “message” commands (see §34.1.2 [Using markers to add links and instructions](#) on page 935):

message URL

message openfile

To specify a target for a **message URL** link (for example, `_blank`):

```
[HTMLOptions]
; URLTarget = name of target to use for all message URL links unless
; otherwise set, default none
URLTarget=_blank
```

To specify an email address:

```
[HTMLOptions]
URLTarget=mailto:name@company.com
```

When a **message openfile** link specifies an absolute path (which must start with a drive letter), **Mif2Go** prefixes the path with “file:///”. For example:

```
message openfile file:///g:/omnisys/ug/out/ugmif2go.pdf
```

For a relative path, **Mif2Go** includes just the text of the destination. For example:

```
message openfile ../out/ugmif2go.pdf
```

Other hypertext “message” commands do not work in HTML, so **Mif2Go** treats them as hypertext **alerts** instead, to bring them to your attention; when you click such a link an “alert” window pops up that displays the contents of the marker as entered in FrameMaker.

20 Providing navigation in HTML

To provide a navigation system for HTML output, you can use **Mif2Go** navigation macros, FrameMaker cross references or hypertext links, or a combination of these, to link together the HTML files **Mif2Go** generates from your FrameMaker document. Topics include:

§20.1 [Understanding how navigation links work](#) on page 627

§20.2 [Generating trails of links](#) on page 627

§20.3 [Including local TOCs](#) on page 631

§20.4 [Creating a browse sequence](#) on page 635

See also:

§19 [Creating HTML links](#) on page 609

20.1 Understanding how navigation links work

Mif2Go navigation features are designed to work in cases where parallel or subordinate headings are in split files of their own, not in the same file as the headings to which they will be linked. This is not a minor point; in fact, the code for local TOCs, trails, and browse sequences depends heavily on the code for file splitting. That is where **Mif2Go** gets the titles and links to use. If you do not split FrameMaker files into topics, those lists of titles and links have no content. For example, a trail link would show only the last *Heading1* in the file, a local TOC would be completely empty, and browse links would go nowhere.

See §18.2 [Splitting files](#) on page 586.

20.2 Generating trails of links

You can have **Mif2Go** generate a trail of links to each topic level in the hierarchy above the current HTML page, and display the trail on each page as an additional navigation aid.

In this section:

§20.2.1 [Understanding trails of links](#) on page 627

§20.2.2 [Specifying whether to include trails of links](#) on page 628

§20.2.3 [Specifying what to include in trails of links](#) on page 628

§20.2.4 [Specifying heading levels for trails of links](#) on page 630

§20.2.5 [Specifying where to display trails of links](#) on page 630

20.2.1 Understanding trails of links

A trail of links, often called a “breadcrumb trail”, typically looks something like this:

[Home & Garden](#) > [Kitchen](#) > [Small appliances](#) > **Coffee makers**

The trail does not necessarily consist of links someone followed to reach a given page; instead, it represents the hierarchical position of the page in the structure of the HTML document.

Each heading in your FrameMaker document that has subheadings can be used as a link in a trail leading to successively lower subheadings. For each trail of links **Mif2Go** inserts the current value of predefined macro `<$_trail>`, which consists of the following:

- Starting HTML code for the trail
- Text of each item in the trail (content of the heading in question)
- Separator code between items in the trail
- Ending code for the trail.

Except for the very last item, a trail of links can include only headings at which file splits occur; see §18.2 [Splitting files](#) on page 586.

Trails of links are not compatible with [HTMLOptions]SmartSplit (see §18.2.2.3 [Preventing splits that leave dangling headings](#) on page 589). If you set SmartSplit=Yes, wherever a heading level is missing the previous heading in the trail will be duplicated. The link will be correct, but the heading text will not.

20.2.2 Specifying whether to include trails of links

To have **Mif2Go** create trails of links, specify the following setting:

```
[Trails]
; MakeTrail = No (default) or Yes (enable use of <$_trail>)
MakeTrail=Yes
```

The default setting, MakeTrail=No, ensures that the overhead of collecting information to construct trails will not be imposed if you do not use this feature.

When MakeTrail=Yes, **Mif2Go** creates a trail for any heading (or other paragraph format) for which both of the following are true:

- The format is assigned the [HTMLParaStyles]Trail and Title properties (see §20.2.3 [Specifying what to include in trails of links](#) on page 628)
- Either of the following is true:
 - The format is assigned the [HTMLParaStyles]Split property (see §18.2.1 [Designating split points](#) on page 586)
 - [Trails]SplitTrail=Yes (see §20.2.5 [Specifying where to display trails of links](#) on page 630).

For both split and extracted files, and for the original file, predefined macro <\$_trail> causes insertion of a trail according to the settings in [Trails]. The trail is always to the first paragraph in the file. **Mif2Go** inserts trails only if MakeTrail=Yes.

20.2.3 Specifying what to include in trails of links

In the usual case, you can set MakeTrail=Yes and SplitTrail=Yes, and appropriate trails of links will appear in the HTML output for headings assigned format properties Split, Trail, and Title. Other settings allow you to override **Mif2Go** defaults for what to include in trails and where to position trails.

Content of trail entries

To include in the trails of links the content of a heading format (or other format, for extracted files), assign both the Title property and the Trail property to that format. For example:

```
[HTMLParaStyles]
; Trail, if [Trails]MakeTrail=Yes, causes the <$_trail> to be put out
; as specified by [Trails] settings.
ChapTitle=Title Trail
FigCaption=ExtrStart Title Trail
```

Heading prefix or suffix

To provide a prefix or suffix for heading content as it appears in trails:

```
[StyleTrailPrefix]
; doc style = prefix to use (if any) for file title in trails
HeadFmt=prefix
```

```
[StyleTrailSuffix]
; doc style = suffix to use (if any) for file title in trails
HeadFmt=suffix
```

The heading content displayed in a trail excludes any prefix or suffix values assigned to heading formats via [StyleTitlePrefix] or [StyleTitleSuffix] (see §18.4.2.3 [Specifying a title prefix or suffix](#) on page 596).

*Code to start,
end, separate
entries*

You can specify the HTML starting, ending, and separator code for the trail. For example:

```
[Trails]
; TrailStart = starting code for <$_trail>
TrailStart=<p><em>
; TrailSep = code between <$_trail> elements
TrailSep=&nbsp;&gt;&nbsp;&nbsp;
; TrailEnd = ending code for <$_trail>
TrailEnd=</em></p>
; TrailLinkClass = value for class attribute in trail links,
; default is none
;TrailLinkClass=trlink
```

You can use TrailStart to put a class attribute on the <p> tag, and use CSS to style it, if the regular italic form (from) does not suffice. And you can add a class attribute for the links used in trails.

Stack trail entries

If the text of your headings tends to be lengthy, you might want to put each item in a trail of links on a separate line, instead of having them all on one line; and you might want to indent each successive entry.

To stack trail entries, replace the final of the TrailSep value with
.

To indent successive stacked entries incrementally, specify the number of spaces to indent; the maximum is four spaces per level:

```
[Trails]
; TrailIndent = number of &nbsp;&nbsp;s to put after TrailSep for each
; output line to create indentation; a value of 1 puts one space
; before the second line, two before the third, three before
; the fourth, etc. A value of 2 puts 2, 4, 6, etc. Zero disables.
; If a value over 4 is set, it is reduced to 4.
TrailIndent=2
```

Current heading

By default, a trail consists of links to headings above the current page. It can also include, as text rather than as a link, the heading (or other first paragraph) of the current page, as the final item in the trail:

```
[Trails]
; TrailCurrent = Yes (default), No, or Always
TrailCurrent=Yes
```

TrailCurrent can have the following values:

- | | |
|--------|---|
| Yes | Include the current-page heading only when the trail already contains at least one item; otherwise omit the trail. This is the default. |
| No | Never include the current-page heading in the trail. |
| Always | Always include the current-page heading, even if it is the only item in the trail. |

You would specify TrailCurrent=Always if, for example, you had embedded another link in TrailStart, perhaps to the table of contents.

20.2.4 Specifying heading levels for trails of links

You can specify the range of heading levels to include in trails of links. The following settings determine the level at which each trail starts and the lowest level where it can end. If headings at a given level are missing from a FrameMaker file, that level is skipped when trails are constructed.

```
[Trails]
; Trail*Level = heading level number
TrailFirstLevel=1
TrailLastLevel=9
```

Normally you would not want trails displayed for headings that do not start new HTML pages, so you would set `TrailLastLevel` to the lowest heading level that actually does start a new page.

You can use the `[TrailLevels]` section to assign a level in the trail to each heading paragraph format. Absent this section, **Mif2Go** uses the levels specified in `[HelpContentsLevels]`. For example:

```
[TrailLevels]
; paraformat = level, from 1 to 9, or 0 to exclude from trails
ChapTitle=1
Heading1=2
Heading1NoNum=2
Heading2=3
```

To exclude a heading format from the trail, you can assign level 0 to that format.

To include a non-heading format as the current (last) item at any level in the trail, you can assign level 9 to that format. For example, if you assign the `[HTMLParaStyles]Trail` property to a paragraph format such as a figure caption that is not part of the hierarchy of headings, and can therefore appear at any level, assign level 9 to that format.

20.2.5 Specifying where to display trails of links

To display a trail of links in each split or extracted file, specify the following setting:

```
[Trails]
; SplitTrail = No (default) or Yes (put <$_trail> out for each split)
SplitTrail=Yes
```

You can specify where the trail of links should appear on each HTML page: before the heading (that is, the first paragraph in the file), after the heading, or at other locations you specify via `[Inserts]`.

```
[Trails]
; TrailPosition = Before (default), After, or Macro
TrailPosition=Before
```

The value of `TrailPosition` determines where the trail appears:

Before	Immediately above the heading (first paragraph). This is the default.
After	Immediately below the heading.
Macro	Wherever you insert predefined macro <code><\$_trail></code> .

When you specify `TrailPosition=Macro`, automatic placement relative to the heading is eliminated, and the trail appears wherever you have included `<$_trail>` in another macro or assigned `<$_trail>` to a specific location.

For example, to show a trail of links at the top of each HTML file and also at the bottom of each split file after the first, you would assign the `<$_trail>` macro to a location

keyword in the [Inserts] section (see §18.5 [Inserting HTML code in split and extract files](#) on page 598):

```
[Inserts]
Top=<$_trail>
SplitBottom=<$_trail>
LastBottom=<$_trail>
```

When **Mif2Go** inserts trails specified by a <\$_trail> macro assigned in the [Inserts] section, if the first paragraph in the file does not have [HTMLParaStyles] property Trail, the trail is not displayed. Otherwise, wherever you assign the <\$_trail> macro, **Mif2Go** inserts a trail according to the settings you specify in the [Trails] section.

20.3 Including local TOCs

You might want to include in each HTML output file a list of links to all files at the next level down: a “local TOC” for subordinate topics, based on the split points you specify (see §18.2 [Splitting files](#) on page 586). For example, if you split FrameMaker files on *Heading1* and *Heading2* paragraphs, at the end of each *Heading1* section you could list links to all *Heading2* paragraphs until the next *Heading1*.

Note: It is not a good idea to use SmartSplit in conjunction with local TOCs; see §18.2.2.3 [Preventing splits that leave dangling headings](#) on page 589.

In this section:

- §20.3.1 [Directing Mif2Go to generate local TOCs](#) on page 631
- §20.3.2 [Configuring local TOCs](#) on page 631
- §20.3.3 [Positioning local TOCs in HTML topics](#) on page 634
- §20.3.4 [Creating local TOCs in FrameMaker](#) on page 635

20.3.1 Directing Mif2Go to generate local TOCs

You can direct **Mif2Go** to create a local TOC automatically for each HTML output file that has subordinate split files. By default, each local TOC consists of links to split files at the next level down; or, you can include all subordinate levels. If an HTML output file has no subordinate files, no local TOC is written.

To direct **Mif2Go** to construct local TOCs:

```
[LocalTOC]
; MakeLocalTOC = No (default) or Yes (enable use of <$_localtoc>)
MakeLocalTOC=Yes
```

This option is required for any other local-TOC settings to take effect.

20.3.2 Configuring local TOCs

In this section:

- §20.3.2.1 [Specifying links to include in local TOCs](#) on page 632
- §20.3.2.2 [Providing HTML code before and after local TOCs](#) on page 632
- §20.3.2.3 [Providing HTML code for each local-TOC entry](#) on page 633
- §20.3.2.4 [Producing multiple-level local TOCs](#) on page 633
- §20.3.2.5 [Overriding default title text for a local-TOC entry](#) on page 634

20.3.2.1 Specifying links to include in local TOCs

To specify which subordinate split files to list in a local TOC, you assign a level number to each of the FrameMaker paragraph formats (usually headings) that designate split points in your document. In other words, you assign a level number to each paragraph format to which you have assigned the [HTMLParaStyles]Split property (see §18.2.1

[Designating split points](#) on page 586):

```
[LocalTOCLevels]
; If this section is missing, [HelpContentsLevels] is used instead.
; Only formats for which [HTMLParaStyles] Split is set are recognized:
; paraformat = level, from 1 to anything; 0 to exclude from local TOC
```

For example, if you split at *Heading1*, *Heading2*, *HeadProc* (which is at the same hierarchical level in your document as *Heading2*), *Heading3*, and *AppxHead1*, you could specify the following settings:

```
[LocalTOCLevels]
Heading1=1
Heading2=2
HeadProc=2
Heading3=3
AppxHead1=0
```

In this example:

- Each *Heading1* split file would have a local TOC consisting of links to all subordinate *Heading2* and *HeadProc* split files.
- Each *Heading2* split file and each *HeadProc* split file that has subordinates would have a local TOC consisting of links to all subordinate *Heading3* split files (if any).
- *AppxHead1* split files would not have local TOCs.

In the [LocalTOCLevels] section, list only paragraph formats that are also split points (have been assigned the [HTMLParaStyles]Split property), and make sure your level number series starts with 1. Otherwise, you might get unwanted blank topics.

If you omit a [LocalTOCLevels] section, **Mif2Go** includes instead any split-point paragraph formats listed in [HelpContentsLevels]. Paragraph formats not listed in either section are ignored.

Note: For any of these settings to take effect, you must enable local TOCs; see §20.3.1 [Directing Mif2Go to generate local TOCs](#) on page 631.

20.3.2.2 Providing HTML code before and after local TOCs

You can specify starting and ending HTML code to surround local TOCs. For example, you might want to provide an introductory phrase before each TOC, and add some space after each TOC.

Mif2Go uses the following default code if you do not specify other values for LocalTOCStart or LocalTOCEnd:

```
[LocalTOC]
; LocalTOCStart = starting code for <$_localtoc>
LocalTOCStart=<p class="localtocstart">In this section:</p>
; LocalTOCEnd = ending code for <$_localtoc>
LocalTOCEnd=<br>
```

To prevent **Mif2Go** from including starting or ending code, you can set either or both options to a blank value:

```
[LocalTOC]
LocalTOCStart=
LocalTOCEnd=
```

Note: For either of these settings to take effect, you must enable local TOCs; see §20.3.1 [Directing Mif2Go to generate local TOCs](#) on page 631.

20.3.2.3 Providing HTML code for each local-TOC entry

You can specify the HTML code to use for local-TOC entries. You can include the following predefined macro variables:

<code><\$_loctocfile></code>	Name of the subordinate file
<code><\$_loctoctype></code>	Title of the subordinate file

These macro variables are effective only when **Mif2Go** produces a local TOC; in any other context they would appear literally, as is usual for undefined macro variables.

Mif2Go uses the following default code if you do not specify another value:

```
[LocalTOC]
; LocalTOCItem = code for each <$_localtoc> item
LocalTOCItem=<p class="localtocitem">\
<a href="<$_loctocfile>"><$_loctoctype></a></p>
```

However, if you specify multiple-level local TOCs, the default is different; see §20.3.2.4 [Producing multiple-level local TOCs](#) on page 633.

The content displayed in a local-TOC item excludes any prefix or suffix value assigned via [StyleTitlePrefix] or [StyleTitleSuffix] (see §18.4.2.3 [Specifying a title prefix or suffix](#) on page 596).

Note: For these macros to take effect, you must enable local TOCs; see §20.3.1 [Directing Mif2Go to generate local TOCs](#) on page 631.

20.3.2.4 Producing multiple-level local TOCs

By default, a local TOC includes links only to split files at the next level down: those that are immediately subordinate to the file where the local TOC appears. To include links to *all* subordinate split files at all levels:

```
[LocalTOC]
; LocalTOCSubs = No (default, include only next level below
; current para) or Yes (include all levels below current para)
LocalTOCSubs = Yes
```

When LocalTOCSubs=Yes, the LocalTOCItem default is changed to begin with:

```
<p class="loctocind<$_loctocind>">
```

where the value of macro variable `<$_loctocind>` is the indentation level of the item, starting with 1.

To increase indentation for each subsequent level you will need a set of CSS classes named `p.loctocind1`, `p.loctocind2`, and so forth, with increasing `margin-left` or `text-indent` values. However, if all links include autonumbers, those might serve as level indicators, and you would not need to provide indentation. In that case, you can define LocalTOCItem as you wish; see §20.3.2.3 [Providing HTML code for each local-TOC entry](#) on page 633.

Note: For this setting to take effect, you must enable local TOCs; see §20.3.1 [Directing Mif2Go to generate local TOCs](#) on page 631.

To provide a multiple-level local TOC only for the first HTML file split from a given FrameMaker file, you can use a list variable in a macro to reset the value of LocalTOCSubs after the first local TOC. For example:

```
[TOCstuff]
<$_localtoc><$$LocalTOC[LocalTOCSubs]=0>
```

See §28.4 [Using multiple-value list variables](#) on page 806.

20.3.2.5 Overriding default title text for a local-TOC entry

Local-TOC links are plain text. By default, the title of the referenced subordinate file becomes the text of each local-TOC link, minus any formatting, embedded links, in-line graphics, and so forth. You can override the default, if you must have formatting in the local-TOC link text, or if you want some other content for the link.

To override the default local-TOC link text, place a custom **LocalTOCTitle** marker in the first paragraph (usually a heading) of the subordinate file, and provide alternate text as the marker content.

See §29.2 [Adding custom marker types](#) on page 832.

20.3.3 Positioning local TOCs in HTML topics

To specify where in an HTML topic a local TOC should appear, assign a predefined local-TOC macro to one of the following:

- a fixed location in each HTML output file
- a location relative to a paragraph format (usually a split-file heading).

The predefined local TOC macros are as follows:

<code><\$_localtoc></code>	A set of links to subordinate topics
<code><\$_lastlocaltoc></code>	A copy of the last local TOC generated

Use predefined macro `<$_lastlocaltoc>` to repeat the last local TOC generated; for example, to provide in each subordinate topic a set of links to all sibling topics.

Note: For either of these macros to take effect, you must enable local TOCs; see §20.3.1 [Directing Mif2Go to generate local TOCs](#) on page 631.

*Local TOC
position in a file*

To specify a position for a local TOC in every HTML output file that should include a local TOC, assign predefined macro `<$_localtoc>` to a location keyword in the `[Inserts]` section. For example, to place a local TOC at the bottom of each split file that has subordinate files:

```
[Inserts]
FirstBottom=<$_localtoc>
SplitBottom=<$_localtoc>
```

See §18.5 [Inserting HTML code in split and extract files](#) on page 598.

*Local TOC
position relative to
a heading*

To specify a position for a local TOC with respect to a paragraph format, assign a local-TOC macro to the format in one of the `[ParaStyleCode*]` sections. For example:

```
[HTMLParaStyles]
Heading1=Split Title CodeAfter
Heading2=Split Title CodeBefore

[ParaStyleCodeAfter]
Heading1=<$_localtoc>

[ParaStyleCodeBefore]
Heading2=<$_lastlocaltoc>
```

Depending on what other local-TOC settings you specify, these assignments would result in the following:

- a list of links to all subordinate *Heading2* files just after each *Heading1* paragraph

- a copy of the list of links to all *Heading2* files subordinate to the same *Heading1*, just before each *Heading2* paragraph (at or near the top of each file split at *Heading2*).

See §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

20.3.4 Creating local TOCs in FrameMaker

You can create local-TOC lists of links in FrameMaker and have **Mif2Go** convert them to HTML, instead of using **Mif2Go** to generate local TOCs. At the end of each section in your document that has subordinate sections, include one of the following:

- a manually inserted list of cross references.
- a generated-TOC text inset.

Mif2Go converts either kind of list to HTML links. The structure of your FrameMaker document would look something like this, with HTML split points indicated:

```

1. Heading1      <----- split here for HTML
    body text . . .
        link to 1.1 Heading2
        link to 1.2 Heading2
1.1 Heading2    <----- split here for HTML
    body text . . .
        link to 1.1.1 Heading3
        link to 1.1.2 Heading3
1.1.1 Heading3  <---- split here for HTML
    body text . . .
1.1.2 Heading3  <---- split here for HTML
    body text . . .
1.2 Heading2    <----- split here for HTML
    body text . . .
2. Heading1      <----- split here for HTML
    body text . . .

```

20.4 Creating a browse sequence

You can use **Mif2Go**-supplied navigation macros to create a browse-type navigation system in HTML, with *previous* and *next* links connecting all files in a single bidirectional series:

<\$_prev> for a link to the preceding HTML file

<\$_next> for a link to the following HTML file.

Mif2Go also provides predefined macro <\$_top> for a jump to top-of-page.

In this section:

- §20.4.1 [Understanding how browse macros work](#) on page 636
- §20.4.2 [Choosing buttons versus text links for a browse sequence](#) on page 638
- §20.4.3 [Formatting browse-link labels](#) on page 639
- §20.4.4 [Modifying macros <\\$_prev>, <\\$_next>, and <\\$_top>](#) on page 639
- §20.4.5 [Understanding browse keyword scope and default values](#) on page 641
- §20.4.6 [Specifying where to invoke a browse macro](#) on page 642
- §20.4.7 [Considering an example of browse navigation](#) on page 643
- §20.4.8 [Specifying an alternate file sequence for browse links](#) on page 644

See also:

- §18 [Splitting and extracting files](#) on page 585
- §19 [Creating HTML links](#) on page 609

§28 [Working with macros](#) on page 787

20.4.1 Understanding how browse macros work

The macros described in this section employ the file sequence information from your FrameMaker .book file to simplify setting up navigation tables at the start and end of HTML pages. **Mif2Go** browse macros use more than one level of indirection to incorporate other macros and macro variables, so that browse links do the right thing in every situation.

In this section:

§20.4.1.1 [Understanding how browse macros vary by file position](#) on page 636

§20.4.1.2 [Understanding how to split files for a browse sequence](#) on page 636

§20.4.1.3 [Understanding equivalent browse macros and macro variables](#) on page 636

§20.4.1.4 [Understanding how browse macros employ macro variables](#) on page 637

20.4.1.1 Understanding how browse macros vary by file position

Default definitions of browse macros `<$_prev>` and `<$_next>`, and the meanings of the macro variables these macros employ, vary according to:

- whether the macros produce text links or buttons (see §20.4.2 [Choosing buttons versus text links for a browse sequence](#) on page 638)
- the sequential position of the HTML file in which the macros are invoked; one of:
 - a split file that is neither the first nor the last split from its FrameMaker file
 - the first file split from any but the first FrameMaker file
 - the last file split from any but the last FrameMaker file
 - the first file split from the first FrameMaker file
 - the last file split from the last FrameMaker file.

You can change any of the definitions by changing the macro code assigned to appropriate navigation keywords; see §20.4.4 [Modifying macros `<\$_prev>`, `<\$_next>`, and `<\$_top>`](#) on page 639.

20.4.1.2 Understanding how to split files for a browse sequence

To create a browse sequence that works across topics split from two or more FrameMaker files, when you direct **Mif2Go** to split FrameMaker files into smaller topics, you must make sure that the very first HTML file split from each FrameMaker chapter actually contains a topic that belongs in the browse sequence. This is always the case when the first paragraph in the FrameMaker file is a heading that you have designated as a split point. Otherwise, if any paragraphs precede the first split-point heading, those paragraphs will end up in an extra HTML file that gets included in the browse sequence, even though it does not contain a topic. You can prevent this from happening with split settings that exclude the unwanted paragraphs; see §18.2.2.2 [Preventing splits that create unwanted files](#) on page 588.

20.4.1.3 Understanding equivalent browse macros and macro variables

For links between book files, **Mif2Go** provides two additional browse macros that are used by indirection: `<$_seqprev>` and `<$_seqnext>`. [Table 20-1](#) shows the default definitions of these macros.

Table 20-1 Indirect navigation macros for files in a book

Macro	Default definition
<code><\$_seqprev></code>	<code><a href="<\$\$_seqprevfile>"><\$\$_seqprevtitle></code>
<code><\$_seqnext></code>	<code><a href="<\$\$_seqnextfile>"><\$\$_seqnexttitle></code>

Because of the equivalences listed in [Table 20-2](#), *you do not need to use* `<$_seqprev>` and `<$_seqnext>`. You can link together all the HTML pages split from all the FrameMaker files in the sequence, using only macros `<$_prev>` and `<$_next>`.

When you are converting a book rather than a single FrameMaker file, for the first split part of any but the very first file in the book, the definition of `<$_prev>` changes to use predefined macro `<$_seqprev>`, which links to the last HTML file split from the previous FrameMaker file in the book. Likewise, for the last HTML file split from any but the last FrameMaker file in the book, the definition of `<$_next>` changes to use predefined macro `<$_seqnext>`, which links to the first HTML file split from the next FrameMaker file in the book. [Table 20-2](#) shows how browse macros and macro variables are equivalent depending on the position of a file in the sequence.

Table 20-2 Equivalent browse macros and variables by file position

File position	Equivalent macros, macro variables, and values
First file split from first FrameMaker file in the sequence	<code><\$_prev></code> = <code><\$_seqprev></code> (produces At Start) <code><\$\$_prevfile></code> = <code><\$\$_seqprevfile></code> = <code><\$\$_seqcurrfile></code> <code><\$\$_prevtitle></code> = <code><\$\$_seqprevtitle></code> = <code><\$\$_seqcurrtitle></code>
First file split from each non-first FrameMaker file, and all non-first unsplit files	<code><\$_prev></code> = <code><\$_seqprev></code> <code><\$\$_prevfile></code> = <code><\$\$_seqprevfile></code> <code><<\$_prevtitle></code> = <code><\$\$_seqprevtitle></code>
Last file split from last FrameMaker file in the sequence	<code><\$_next></code> = <code><\$_seqnext></code> (produces At End) <code><\$\$_nextfile></code> = <code><\$\$_seqnextfile></code> = <code><\$\$_seqcurrfile></code> <code><\$\$_nexttitle></code> = <code><\$\$_seqnexttitle></code> = <code><\$\$_seqcurrtitle></code>
Last file split from each non-last FrameMaker file, and all non-last unsplit files	<code><\$_next></code> = <code><\$_seqnext></code> <code><\$\$_nextfile></code> = <code><\$\$_seqnextfile></code> <code><<\$_nexttitle></code> = <code><\$\$_seqnexttitle></code>

20.4.1.4 Understanding how browse macros employ macro variables

The definition of each browse macro includes predefined macro variables for a destination for the link and for a label. [Table 20-3](#) shows the default values of the link destination and link label used in `<$_prev>` and `<$_next>` for each file position.

Table 20-3 Default destination and label values for browse macros

Macro	File position in sequence	Destination value	Label value
<code><\$_prev></code>	First file split from first FrameMaker file in sequence	<i>None (no destination code)</i>	At Start
	First file split from: – each non-first FrameMaker file – all non-first unsplit files	<code><\$\$_seqprevfile></code>	<code><\$\$_seqprevtitle></code>
	All other split files	<code><\$\$_prevfile></code>	<code><\$\$_prevtitle></code>

Table 20-3 Default destination and label values for browse macros

Macro	File position in sequence	Destination value	Label value
<\$_next>	Last file split from last FrameMaker file in sequence	<i>None (no destination code)</i>	At End
	Last file split from: - each non-last FrameMaker file - all non-last unsplit files	<\$\$_seqnextfile>	<\$\$_seqnexttitle>
	All other split files	<\$\$_nextfile>	<\$\$_nexttitle>

Table 20-4 shows the meanings of the macro variables used in browse-macro definitions.

Table 20-4 Component macro variables for browse macros

Macro	Macro variable	Description
<\$_prev>	<\$\$_prevfile>	File name of preceding split file
	<\$\$_prevtitle>	Title of preceding split file
	<\$\$_seqprevfile>	Name of preceding file in sequence, with extension .htm
	<\$\$_seqprevtitle>	Title listed in sequence for preceding FrameMaker file
<\$_next>	<\$\$_nextfile>	File name of following split file
	<\$\$_nexttitle>	Title of following split file
	<\$\$_seqnextfile>	Name of following file in sequence, with extension .htm
	<\$\$_seqnexttitle>	Title listed in sequence for following FrameMaker file
<i>Either macro</i>	<\$\$_currfile>	File name of current split file
	<\$\$_currtitle>	Title of current split file (the one used in HTML <title> element)
	<\$\$_seqcurrfile>	Name of current file in sequence, with extension .htm
	<\$\$_seqcurrtitle>	Title listed in sequence for current FrameMaker file

See §18.6 [Referencing split and extract files](#) on page 600 for additional macro variables that refer to file names and titles of split and extracted files.

20.4.2 Choosing buttons versus text links for a browse sequence

By default, **Mif2Go** navigation macros <\$_prev> and <\$_next> produce simple text links. However, you can specify buttons instead:

```
[NavigationMacros]
; UseNavButtons = No (default, use links for <$_prev> and <$_next>)
; or Yes (change the set of defaults to those for buttons instead)
UseNavButtons = Yes
```

When UseNavButtons=Yes, **Mif2Go** navigation macros produce JavaScript code such as the following:

```
<button type="button"
  onclick="javascript:location.href='Destination'">Label</button>
```

When UseNavButtons=No, **Mif2Go** navigation macros produce HTML code such as this:

```
<a href="Destination">Label</a>
```

Both button and text-link navigation macros use predefined macro variables to provide appropriate values for *Destination* and *Label*; see [Table 20-3](#) on page 637.

20.4.3 Formatting browse-link labels

You can provide formatting for *Label* content wherever you invoke text-link `<$_prev>` and `<$_next>` macros. For example:

```
<p class="navlabel"><$_prev></p>
```

Button labels are at the mercy of default browser rendering, unless you redefine `<$_prev>` and `<$_next>` macros so they include formatting. For buttons, formatting tags must be placed *within* the `<button>` tag, surrounding *Label* text. For example (must be all on one line):

```
<button type="button"
  onclick="javascript:location.href='somefile.htm'">
  <p class="navlabel">Previous</p></button>
```

20.4.4 Modifying macros `<$_prev>`, `<$_next>`, and `<$_top>`

With care, you can redefine any of the browse macros to include formatting, alternate link destinations, or alternate labels. **Mif2Go** provides keywords to which you assign macro code for this purpose.

In this section:

§20.4.4.1 [Redefining text-link browse macros](#) on page 639

§20.4.4.2 [Redefining button browse macros](#) on page 640

20.4.4.1 Redefining text-link browse macros

When `UseNavButtons=No` (the default) you can redefine browse macros for text links by changing the code assigned to the keywords listed in this section.

Do not try to duplicate `<$_prev>` and `<$_next>` logic by using the predefined macro variable components outside of the definitions for `<$_prev>` and `<$_next>`. The browse sequence would fail on inter-file links, because you would be missing some critical internal code that is required to handle such links.

Note: In your configuration file each code assignment must be all on one line, even if it does not look that way here.

To redefine macros for text links between HTML files split from a FrameMaker file, change the following default definitions:

```
[NavigationMacros]
; PrevMacro = content to put out for <$_prev>
PrevMacro = <a href="<$$_prevfile>"><$$_prevtitle></a>
; NextMacro = content to put out for <$_next>
NextMacro = <a href="<$$_nextfile>"><$$_nexttitle></a>
```

For a *previous* text link in the first and a *next* text link in the last HTML file split from a FrameMaker file, change the following default definitions:

```
[NavigationMacros]
; PrevFSMacro = <$_seqprev>, macro to use for <$_prev>
; at start of file
PrevFSMacro= <a href="<$$_seqprevfile>"><$$_seqprevtitle></a>
; NextFSMacro = <$_seqnext>, macro to use for <$_next> at end of file
NextFSMacro= <a href="<$$_seqnextfile>"><$$_seqnexttitle></a>
```

For a *previous* text “link” in the first and a *next* text “link” in the last HTML file in the entire sequence, change the following default definitions:

```
[NavigationMacros]
; StartingPrevFSMacro = <$_prev> to use at start of first file in book
```

```
StartingPrevFSMacro = <$$seqstarttitle>
; EndingNextFSMacro = <$_next> to use at end of last file in book
EndingNextFSMacro = <$$seqendtitle>
```

For a text link to the top of the current page, change the following default definition:

```
[NavigationMacros]
; TopMacro = content to put out for <$_top>,
; link to top of current file
TopMacro = <a href="<$$currfile>"><$$toptitle></a>
```

To change a macro definition, modify or replace the macro code assigned to the appropriate keyword. For example, to add a title attribute to the <\$_prev> link for WAI purposes (the definition must be all on the same line):

```
PrevMacro =
<a href="<$$prevfile>" title="<$$prevtitle>"><$$prevtitle></a>
```

For the first and last HTML files in the entire sequence, <\$_prev> and <\$_next> do not use links at all, but only label content. If you want static labels for the other links, you could redefine the text-link macros as follows:

```
[NavigationMacros]
PrevMacro = <a href="<$$prevfile>">Prev</a>
NextMacro = <a href="<$$nextfile>">Next</a>
PrevFSMacro = <a href="<$$prevfile>">Prev</a>
NextFSMacro = <a href="<$$nextfile>">Next</a>
```

To make the first and last links to reference places outside the document (for example):

```
[NavigationMacros]
StartingPrevFSMacro = <a href="<$$HomeURL>">Home</a>
EndingPrevFSMacro = <a href="<$$Plan2URL>">Plan 2</a>

[MacroVariables]
HomeURL = http://www.oursite.org/index.htm
Plan2URL = http://www.oursite.org/greatplans/plan2.htm
```

20.4.4.2 Redefining button browse macros

When UseNavButtons=Yes, you can redefine browse macros for buttons by changing the code assigned to the keywords listed in this section.

Do not try to duplicate <\$_prev> and <\$_next> logic by using the predefined macro variable components outside of the definitions for <\$_prev> and <\$_next>. The browse sequence would fail on inter-file links, because you would be missing some critical internal code that is required to handle such links.

Note: In your configuration file each code assignment must be all on one line, even if it does not look that way here.

To redefine macros for buttons that activate links between the HTML files split from a FrameMaker file, change the following default definitions:

```
[NavigationMacros]
UseNavButtons = Yes
; PrevButton = content to put out for <$_prev>
PrevButton = <button type="button"
onclick="javascript:location.href='<$$prevfile>'">
<$$prevtitle></button>
; NextButton = content to put out for <$_next>
NextButton = <button type="button"
onclick="javascript:location.href='<$$nextfile>'">
<$$nexttitle></button>
```

For a *previous* button in the first and a *next* button in the last HTML file split from a FrameMaker file, change the following default definitions:

```
[NavigationMacros]
UseNavButtons = Yes
; PrevFSButton = <$_seqprev>, macro to use for <$_prev>
; at start of file
PrevFSButton = <button type="button"
  onclick="javascript:location.href='<$$_seqprevfile>'">
  <$$_seqprevtitle></button>
; NextFSButton = <$_seqnext>, macro to use for <$_next> at end of file
NextFSButton = <button type="button"
  onclick="javascript:location.href='<$$_seqnextfile>'">
  <$$_seqnexttitle></button>
```

For a *previous* button in the first and a *next* button in the last HTML file in the entire sequence, change the following default definitions:

```
[NavigationMacros]
UseNavButtons = Yes
; StartingPrevFSButton = <$_prev> to use at start of first file
; in book
StartingPrevFSButton = <button type="button">
  <$$_seqstarttitle></button>
; EndingNextFSButton = <$_next> to use at end of last file in book
EndingNextFSButton = <button type="button">
  <$$_seqendtitle></button>
```

For a button link to the top of the current page, change the following default definition:

```
[NavigationMacros]
UseNavButtons = Yes
; TopButton = content to put out for <$_top>,
; link to top of current file
TopButton = <button type="button"
  onclick="javascript:location.href='<$$_currfile>'">
  <$$_toptitle></button>
```

To provide label formatting (for example):

```
[NavigationMacros]
UseNavButtons = Yes
PrevButton = <button type="button"
  onclick="javascript:location.href='<$$_prevfile>'">
  <p class="navcell">Prev</p></button>
```

To provide different text for the (non-link) very first and very last buttons, or no text at all:

```
[NavigationMacros]
UseNavButtons = Yes
StartingFSButton =
EndingFSButton = <b>Stop!</b>
```

Typing nothing (or only a single space) after the equals sign results in a null value.

20.4.5 Understanding browse keyword scope and default values

Table 20-5 shows the scope of each browse-macro keyword with respect to file position, the macros each keyword defines, and the macro variables used in each default value.

Table 20-5 Scope of [NavigationMacros] keywords

Scope	Keyword (* = Macro or Button)	Default value uses:		
		Defines:	File name	Label
Very first file	StartingPrevFS*	<i>None</i>	<i>None</i>	At Start
Very last file	EndingNextFS*	<i>None</i>	<i>None</i>	At End
First split	PrevFS*	<\$_seqprev>	<\$\$_seqprevfile>	<\$\$_seqprevtitle>
Last split	NextFS*	<\$_seqnext>	<\$\$_seqnextfile>	<\$\$_seqnexttitle>
All other split files	Prev*	<\$_prev>	<\$\$_prevfile>	<\$\$_prevtitle>
	Next*	<\$_next>	<\$\$_nextfile>	<\$\$_nexttitle>

Table 20-6 shows the default value Mif2Go uses for each browse-macro keyword when the navigation macros produce text links; Table 20-7 shows the default values when macros produce buttons (see §20.4.2 [Choosing buttons versus text links for a browse sequence](#) on page 638).

Table 20-6 Default values of text-link browse keywords

Keyword	Default value
PrevMacro	<a href="<\$\$_prevfile>"><\$\$_prevtitle>
NextMacro	<a href="<\$\$_nextfile>"><\$\$_nexttitle>
PrevFSMacro	<a href="<\$\$_seqprevfile>"><\$\$_seqprevtitle>
NextFSMacro	<a href="<\$\$_seqnextfile>"><\$\$_seqnexttitle>
StartingPrevFSMacro	At Start
EndingNextFSMacro	At End

Table 20-7 Default values of button browse keywords

Keyword	Default value
PrevButton	<button type="button" onclick="javascript:location.href='<\$\$_prevfile>'"> <\$\$_prevtitle></button>
NextButton	<button type="button" onclick="javascript:location.href='<\$\$_nextfile>'"> <\$\$_nexttitle></button>
PrevFSButton	<button type="button" onclick="javascript:location.href='<\$\$_seqprevfile>'"> <\$\$_seqprevtitle></button>
NextFSButton	<button type="button" onclick="javascript:location.href='<\$\$_seqnextfile>'"> <\$\$_seqnexttitle></button>
StartingPrevFSButton	<button type="button">At Start</button>
EndingNextFSButton	<button type="button">At End</button>

20.4.6 Specifying where to invoke a browse macro

To specify where in an HTML output file to invoke one or more browse macros, in the [Inserts] section assign the macro(s) to one or both of the following keywords:

Top	At the beginning of the <body> element.
Bottom	Just before the end of the <body> element.

For example:

```
[Inserts]
Top = <$_prev><br /><br /><$_next><br /><br />
Bottom = <$_top>
```

See §18.5 [Inserting HTML code in split and extract files](#) on page 598 for additional [Inserts] keywords you can use to specify other file locations, and for keyword prefixes you can use to restrict macro assignment by output file type.

20.4.7 Considering an example of browse navigation

Suppose your project involves a FrameMaker document named `TechGuide.book` that consists of three files: `Intro.fm`, `Examples.fm`, and `Summary.fm`; and suppose you want **Prev** and **Next** links at the top of each HTML file generated from `TechGuide.book`.

In the [Inserts] section of the configuration file you would specify where on each page of HTML output the links should appear (see §20.4.6 [Specifying where to invoke a browse macro](#) on page 642). For Examples, at the top of each HTML page:

```
[Inserts]
Top = <$_prev><br /><br /><$_next>
```

Suppose **Mif2Go** splits the files in `TechGuide.book` as follows (see §18.2 [Splitting files](#) on page 586):

```
Intro.fm:  Intro.htm  aa100002.htm  aa100003.htm
Examples.fm: Examples.htm  bb200002.htm  bb200003.htm  bb200004.htm
Summary.fm: Summary.htm  cc300002.htm
```

[Figure 20-1](#) shows the positions of these files with respect to the differences in default definitions of `<$_prev>` and `<$_next>` (see [Table 20-3](#) on page 637).

Figure 20-1 Positions of files in TechGuide.book

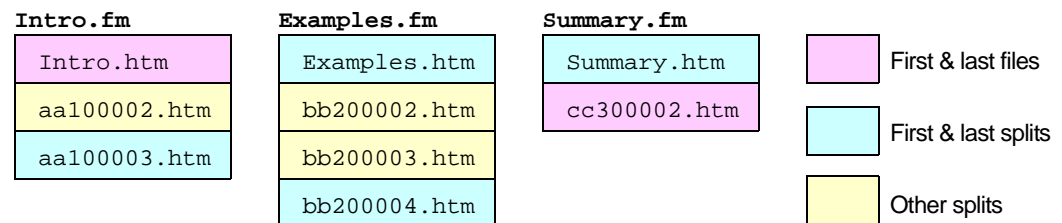


Table 20-8 Values of variables in navigation links for TechGuide.book

HTML file	<\$_prev>		<\$_next>	
	<\$\$_prevfile>	<\$\$_prevtitle>	<\$\$_nextfile>	<\$\$_nexttitle>
Intro.htm	None (no link)	At Start	aa100002.htm	Intro section 2 title
aa100002.htm	Intro.htm	Introduction	aa100003.htm	Intro section 3 title
aa100003.htm	aa100002.htm	Intro section 2 title	Examples.htm	Brilliant examples and exposition
Examples.htm	Intro.htm	Introduction	bb200002.htm	Examples section 2 title
bb200002.htm	Examples.htm	Brilliant examples and exposition	bb200003.htm	Examples section 3 title
bb200003.htm	bb200002.htm	Examples section 2 title	bb200004.htm	Examples section 4 title

Table 20-8 Values of variables in navigation links for TechGuide.book

HTML file	<\$ _prev>		<\$ _next>	
	<\$\$ _prevfile>	<\$\$ _prevtitle>	<\$\$ _nextfile>	<\$\$ _nexttitle>
bb200004.htm	bb200003.htm	<i>Examples section 3 title</i>	Summary.htm	Conclusion
Summary.htm	Examples.htm	Brilliant examples and exposition	cc300002.htm	<i>Summary section 2 title</i>
cc300002.htm	Summary.htm	Conclusion	<i>None (no link)</i>	At End

Table 20-8 shows the values that <\$ _prev> and <\$ _next> macros would use in each HTML file; in particular:

- For the very first and very last files, the default definitions of <\$ _prev> and <\$ _next> do not include links, but only predefined text for titles.
- For all other files, the title is that used in the HTML <title> element, and usually comes from a paragraph or a FrameMaker marker, as determined by settings described in §18.4.2 [Specifying page titles for split or extract files](#) on page 594.

20.4.8 Specifying an alternate file sequence for browse links

If you are converting a FrameMaker book, the browse sequence includes all files in the book. To exclude one or more FrameMaker files from the browse sequence, or to change the order in which files are linked, you must create an alternate FrameMaker book that contains the files you want in the order you want.

Note: The former [FileSequence] section is deprecated, and is no longer used by Mif2Go.

21 Mapping text formats to HTML/XML

This section shows how to assign HTML elements to FrameMaker paragraph and character formats. Topics include:

- §21.1 [Understanding how Mif2Go converts text](#) on page 645
- §21.2 [Choosing how to map formats](#) on page 645
- §21.3 [Mapping paragraph formats](#) on page 646
- §21.4 [Mapping character formats](#) on page 653
- §21.5 [Assigning properties to text formats](#) on page 653
- §21.6 [Mapping special characters](#) on page 658
- §21.7 [Mapping fonts](#) on page 663
- §21.7 [Mapping fonts](#) on page 663
- §21.8 [Managing typographic elements for HTML or XML](#) on page 667
- §21.9 [Specifying text colors for HTML](#) on page 669
- §21.10 [Configuring preformatted text for HTML/XML](#) on page 670
- §21.11 [Converting footnotes to HTML or XML](#) on page 671
- §21.12 [Converting list formats to HTML](#) on page 674

See also:

- §22 [Setting up CSS for HTML](#) on page 681

21.1 Understanding how Mif2Go converts text

- | | |
|--------------------------------------|--|
| <i>Format overrides included</i> | By default, Mif2Go writes tags for all FrameMaker text properties, even for overrides. This preserves text appearance in HTML. However, you can provide configuration settings that disallow all or selected overrides, for all or selected formats; see §21.5 Assigning properties to text formats on page 653. |
| <i>Master-page text not included</i> | Mif2Go does not convert master page headers and footers at all for HTML; you hardly ever want the same page top and bottom as in print files. However, you can specify predetermined HTML to appear at the top and bottom of output pages; see §28.9.2 Invoking macros at predetermined points in output on page 821. |

21.2 Choosing how to map formats

Mif2Go provides several ways to map FrameMaker formats to HTML, with considerable overlap among methods. You might want to use some or all of the following:

- [Conversion template](#)
- [Configuration settings](#)
- [Cascading style sheets](#)

- | | |
|-------------------------------|--|
| <i>Conversion template</i> | Import formats from a FrameMaker template you design specifically to produce HTML that looks the way you want. When you import formats from an alternate FrameMaker template, you can control text appearance; you can also redefine cross-reference formats to remove page numbers, which is not possible with other methods. See: <ul style="list-style-type: none">§2.4 Importing formats from a conversion template on page 67§30.7 Applying FrameMaker conversion templates on page 863. |
| <i>Configuration settings</i> | Insert settings in a configuration file to map paragraph and character formats individually to HTML tags. The display attributes of HTML tags to which you map individual formats |

are browser dependent. All you can be sure of is that, by default (without CSS), an h1 will look “bigger” than an h2, and so forth. Sometimes the “biggerness” is questionable, especially at the lower end (h4, h5, h6). However, you can use configuration settings to achieve effects not possible with CSS or with a FrameMaker template, such as the macro insertion of content. See:

§21.3 [Mapping paragraph formats](#) on page 646

§21.4 [Mapping character formats](#) on page 653.

Cascading style sheets

Use cascading style sheets (CSS). Using CSS might cause different effects in different browsers, or even in different versions of the same browser. However, you can override CSS with individual settings in the configuration file. See §22 [Setting up CSS for HTML](#) on page 681.

Note: Any formatting that is directly created by an HTML tag overrides CSS. Using HTML presentational tags and attributes cripples your ability to use CSS, and therefore to adjust formatting easily without having to alter content.

21.3 Mapping paragraph formats

By default, if you do not explicitly map FrameMaker formats to HTML tags, **Mif2Go** does the following:

- Uses <p> as the tag for all paragraph formats.
- Treats all character formats as overrides.
- Creates tags for all format properties, including overrides.
- Converts all tag names to valid CSS names, without spaces or non-alphanumeric characters, leading digits, or accented characters (the latter become unaccented).

This might be adequate, especially if you are using CSS. However, you might want your headings to come out with <h1> styles, your emphasized text to be tagged , and your lists to become real HTML indented lists, without requiring CSS.

In this section:

§21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646

§21.3.2 [Converting sidehead and run-in paragraph formats](#) on page 648

§21.3.4 [Including text-frame content in line](#) on page 649

§21.3.5 [Designating script paragraph formats](#) on page 650

§21.3.6 [Stripping paragraph properties](#) on page 650

§21.3.7 [Keeping or removing reference frames](#) on page 651

§21.3.8 [Deciding how to treat forced returns](#) on page 651

§21.3.9 [Providing content for empty paragraphs](#) on page 651

§21.3.10 [Eliminating empty paragraphs in text](#) on page 652

§21.3.11 [Eliminating invisible paragraphs](#) on page 652

§21.3.12 [Eliminating unwanted paragraphs](#) on page 652

See also:

§21.10 [Configuring preformatted text for HTML/XML](#) on page 670

§21.11 [Converting footnotes to HTML or XML](#) on page 671

§21.12 [Converting list formats to HTML](#) on page 674

21.3.1 Assigning HTML tags and attributes to paragraph formats

To specify the HTML tag to be used for headings and other special-purpose formats:

```
[ParaTags]
; Document para format name = HTML style name (default is <p>)
; use h1-h6, pre, script, address, or blockquote for HTML styles
```

Although these (and `span`) are the only valid HTML tag names you can specify in this section, **Mif2Go** does not require you to stick to valid tags. You can use any tags, to allow XML within HTML. In fact, you can use any text that can go inside the `<>` brackets **Mif2Go** supplies around the text. However, only tags valid in HTML for paragraphs produce effects in HTML output.

If you are creating Web pages that will be available to search engines, keep in mind that headings that are actually tagged as headings (h1 through h6) can be important for search ranking. For example, Google search might look at the following (rather than keywords in meta tags):

1. titles of pages
2. words displayed in links to those pages
3. words used in headings that are tagged as such
4. words used within the pages.

With paragraph tag settings in `[ParaTags]` you can also do the following:

[Add attributes to a tag](#)

[Apply a character tag to a paragraph format](#)

[Provide a CSS class name](#)

[Suppress paragraph tags entirely.](#)

*Add attributes to
a tag*

To add attributes to the paragraph tag, list them after the tag. For example:

```
[ParaTags]
CodeBold = pre type="bold"
```

Everything after the first space that follows the tag name is removed for the end tag. To apply an attribute to an individual instance of a paragraph format, insert an attribute marker in the paragraph; see §29.2.4 [Using attribute markers for HTML or XML](#) on page 835. For this example, you would use a marker of type **ParaType** with content **bold** (no quotation marks).

*Apply a character
tag to a
paragraph format*

To apply an HTML character tag (for example, `em`) to a paragraph format, you would have to do something like this:

```
[HTMLParaStyles]
ParaFmt = CodeStart CodeEnd

[ParaStyleCodeStart]
ParaFmt = <em>

[ParaStyleCodeEnd]
ParaFmt = </em>
```

(With CSS, it might be simpler to add `font-style: italic;` to the CSS style for the `<p.parafmt>` tag.)

*Provide a CSS
class name*

If you are using CSS, by default the tag name becomes the CSS class name for HTML output; for XML output, the default is reversed. See §22.5 [Understanding how CSS affects other options](#) on page 687.

You can provide your own class names. For example:

```
[ParaTags]
Heading 1 = H1 class="tophead"
```

results in:

```
<h1 class="tophead">
```

for all *Heading 1* paragraphs in HTML output.

If you do provide your own class names, do not also have **Mif2Go** generate a style sheet (see §22.4.1 [Specifying CSS options at project set-up time](#) on page 683); you would get duplicate class entries for any such paragraph formats. See also §22.7.2 [Mapping paragraph formats to CSS classes](#) on page 692.

For XML output, see §14.4.2 [Deriving XML tags from format and class names](#) on page 462.

*Suppress
paragraph tags
entirely*

To eliminate style tags entirely, map the paragraph format to nothing:

```
[ParaTags]
ParaFmt =
```

Specifying an empty [ParaTags] class is equivalent to assigning format property NoPara to the paragraph format; see §21.3.6 [Stripping paragraph properties](#) on page 650.

If you are producing DITA XML output, see also §15.4.3.2 [Omitting element tags for selected paragraph formats](#) on page 488.

21.3.2 Converting sidehead and run-in paragraph formats

Mif2Go does not provide any special treatment for sideheads in HTML. A sidehead paragraph appears in HTML output as a normal paragraph.

HTML does not support run-in formats. By default, **Mif2Go** tries to duplicate the FrameMaker run-in effect in HTML by converting a run-in heading to use the paragraph format and CSS class of the run-in body paragraph (the paragraph the heading runs into), modified by the character formatting, first-line indent, and space-before of the run-in heading.

To have **Mif2Go** instead place the run-in heading on a line previous to the following paragraph, preserving all paragraph and character properties of the heading:

```
[HTMLOptions]
; RunInHeads = Runin (default) or Normal (head on previous line,
; default for DITA output)
RunInHeads = Normal
```

To have **Mif2Go** base the HTML tag used for the combined paragraphs on the paragraph format name of the run-in head, instead of on the format of the following paragraph:

```
[HTMLOptions]
; UseRunInTag = No (default, get tag from body para)
; or Yes (from runin)
UseRunInTag = Yes
```

For example, if *Heading4* is a run-in heading followed by a *Body* paragraph, to combine both into an HTML heading:

```
[HTMLOptions]
UseRunInTag = Yes

[ParaTags]
Heading4 = H1
```

21.3.3 Converting paragraph formats with autonumbers

By default, for HTML output **Mif2Go** omits autonumber characters from paragraph formats that are mapped to HTML list styles, and converts autonumbers to text for all other paragraph formats. For XML output, the default is to omit all autonumbers.

To eliminate autonumbers from selected paragraph formats:

```
[HTMLParaStyles]
; NoAnum excludes autonumber in non-list items, default keeps it
; for formats other than DocBook and DITA.
ParaFmt = NoAnum
```

For example, to eliminate bullets:

```
[HTMLParaStyles]
Bulleted = NoAnum
```

To eliminate autonumbers from all paragraph formats:

```
[HTMLOptions]
; UseAnums = Yes (HTML default, use unless list type)
; or No (XML default)
UseAnums = No
```

To override UseAnums=No for selected paragraph formats:

```
[HTMLParaStyles]
; Anum includes Frame autonumber in para. For lists, and for
; DocBook and DITA, the default omits it.
ParaFmt = Anum
```

To override UseAnums=Yes for lists in FrameMaker generated files, apply a unique character format (for example, *ListAnum*) to the <paranum> part of each entry on the reference page for the generated file; and include the following setting:

```
[HTMLParaStyles]
ListAnum = Delete
```

To eliminate unwanted tabs from paragraph autonumbers:

```
[HTMLOptions]
; AnumTabs = Yes (default, make tab in numbering properties into space
; unless in [HTMLParaStyles] List format, in which case remove it)
; or No (remove)
AnumTabs = Yes
```

See also:

§21.6.2 [Understanding how Mif2Go treats tabs in HTML/XML](#) on page 658

§21.12 [Converting list formats to HTML](#) on page 674

§34.7 [Converting autonumbers for database systems](#) on page 944

21.3.4 Including text-frame content in line

If your FrameMaker document includes text frames inside anchored frames (for example, for sidebars or notes), you might want the text in those frames to appear in line with the main flow in HTML.

To include text-frame content in line:

```
[HTMLParaStyles]
; TextFrameIsText is applied to an anchor para format to cause
; anchored frames containing a text frame to be rendered as in-line
; text.
AnchorParaFmt = TextFrameIsText
```

Use `TextFrameIsText` to include the content of a separate FrameMaker text frame (a text frame within an anchored frame) directly in line with the body text where the frame is anchored. Assign `TextFrameIsText` to the paragraph format that contains the anchor. If you are using a dedicated paragraph format for anchors, also assign `NoPara` and `NoTags` to the format (see §21.3.6 [Stripping paragraph properties](#) on page 650) to eliminate the empty anchor paragraph. For example:

```
[HTMLParaStyles]
SideBarAnchor = TextFrameIsText NoPara NoTags
```

21.3.5 Designating script paragraph formats

A paragraph tagged as `script` in `[ParaTags]` (see §21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646) includes a type attribute that is always added in the opening tag:

```
[HTMLOptions]
; ScriptType = text/javascript (default) or other MIME type
ScriptType = text/javascript
```

If you apply the `[HTMLParaStyles]Comment` attribute to such a paragraph, the script body begins and ends with automatic comment delimiters. See §21.3.6 [Stripping paragraph properties](#) on page 650.

21.3.6 Stripping paragraph properties

You can designate text in your FrameMaker document that you do not want fully converted to HTML. For example, you can include material pre-written in HTML, and direct **Mif2Go** to insert the material *as is* in the HTML output. Create a special paragraph format to use only for this purpose, and assign to it one of the following properties:

```
[HTMLParaStyles]
; para format = keywords for functions and properties
; Comment makes the element a comment, replacing the para tags,
;   unless ParaStyle is "script", then the comment is in the tags
; NoTags suppresses any attributes for the para tag, and suppresses
;   any tags within
; NoPara eliminates the para tags only, to be provided in a macro
; NoWrap suppresses \n line breaks and preserves leading spaces
; Raw acts like NoTags, and also eliminates the para tags entirely
;   It is used to put macro inclusions in between document elements
```

- | | |
|----------------|--|
| <i>Comment</i> | Use <code>Comment</code> to cause a paragraph to appear only as a comment in the generated HTML source code. Mif2Go substitutes <code><!--</code> and <code>--></code> tags for the <code><p></code> and <code></p></code> tags, unless you have also assigned a <code>script</code> tag to the paragraph format in <code>[ParaTags]</code> ; see §21.3.5 Designating script paragraph formats on page 650. |
| <i>NoTags</i> | Use <code>NoTags</code> to suppress all tags between <code><p></code> and <code></p></code> (such as <code></code> , <code></code> , <code><i></code> , and so forth) in the generated HTML for the paragraph. Only the <code><p></code> tags themselves and the paragraph content are included in the output. |
| <i>NoPara</i> | Use <code>NoPara</code> to suppress only the <code><p></code> tags in the output. You might want to do this when either of the following is true: <ul style="list-style-type: none"> • The paragraph will be part of a Mif2Go macro that already supplies <code><p> ... </p></code>. • You are generating XML instead of HTML; see §14.4.3 Eliminating HTML attributes and tags for generic XML on page 463. If you are producing DITA XML output, also see §15.4.3.2 Omitting element tags for selected paragraph formats on page 488. |
| <i>NoWrap</i> | Use <code>NoWrap</code> to suppress <code>\n</code> line breaks and preserve leading spaces in preformatted text. This property has the same effect as <code>[HTMLOptions]NoWrap</code> , but applied at the paragraph format level; see §13.6.4 Suppressing line breaks in HTML and XML output on page 437. |
| <i>Raw</i> | Use <code>Raw</code> to insert straight HTML code wherever you want it to appear in your document. Mif2Go embeds the content of the paragraph in the output without generating HTML tags, and without processing any macro invocations the content might include. |

21.3.7 Keeping or removing reference frames

Sometimes you want the *Frame Above* and *Frame Below* specified with your paragraph formats to be converted, and sometimes not. For DITA XML output, most likely you do not want them converted.

To set the default behavior to remove all reference frames:

```
[HTMLOptions]
; RemoveFramesAbove = No (default) or Yes (can be overridden)
RemoveFramesAbove = Yes
; RemoveFramesBelow = No (default) or Yes (can be overridden)
RemoveFramesBelow = Yes
```

You can override the default, format by format:

```
[HTMLParaStyles]
; FrameAbove and NoFrameAbove override the RemoveFramesAbove default
ChapHead = NoFrameAbove
; FrameBelow and NoFrameBelow override the RemoveFramesBelow default
ChapClose = NoFrameBelow
```

Reference frames must be in a usable format: imported GIFs or JPEGs, or images produced by the FrameMaker graphic export filter.

See also:

- §23.5.4 [Converting reference-page graphics for HTML](#) on page 712
- §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883
- §31.2.5.7 [Converting graphics on reference pages](#) on page 885

21.3.8 Deciding how to treat forced returns

Except within preformatted text, by default **Mif2Go** converts a forced return (FrameMaker **Shift+Enter**, a typesetting “hard return”) to one of the following, depending on the output type:

```
HTML:    <br>
XHTML:   <br />
XML:     a space
```

For HTML and XHTML output only, to override the default for selected paragraph formats:

```
[HTMLParaStyles]
; NoBreak changes any Shift+Enter in the para to space instead of
; <br> for HTML and XHTML.
ParaFmt = NoBreak
```

The NoBreak property has no effect in `<pre>` where a forced return becomes a line break, nor in those XML formats where `
` is invalid, such as DITA and DocBook.

See also:

- §13.6.4 [Suppressing line breaks in HTML and XML output](#) on page 437
- §14.4.5 [Configuring forced returns for XML](#) on page 465
- §21.10 [Configuring preformatted text for HTML/XML](#) on page 670

21.3.9 Providing content for empty paragraphs

To specify text content for paragraphs that are otherwise blank (empty):

```
[HTMLOptions]
; EmptyParaContent = string to put in otherwise-empty paragraphs
EmptyParaContent = &nbsp;
```

A single nonbreaking space is the default.

Note: Setting `EmptyParaContent=0` (zero) inserts a literal “0”: a string, not a number.

21.3.10 Eliminating empty paragraphs in text

If a paragraph is empty, it usually takes up more space in HTML than it did in FrameMaker, often three times as much. To eliminate empty paragraphs:

```
[HTMLOptions]
; RemoveEmptyParagraphs = No (default) or Yes (remove paras w/o text)
RemoveEmptyParagraphs = Yes
```

When `RemoveEmptyParagraphs=Yes`, tags for empty paragraphs in text are not included in HTML output. This setting does not affect either of the following:

- empty paragraphs in table cells, for which there is a separate configuration setting; see §24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744
- preformatted text, where empty paragraph tags are always preserved.

Mif2Go considers a paragraph that contains only markers (no text) to be empty for the purpose of generating HTML tags. Therefore when `RemoveEmptyParagraphs=Yes`, the markers still work, even though the paragraph itself disappears. However, any macro code you assign to the paragraph format will *not* be executed for empty instances of that format.

See also:

§23.5.7 [Retaining run-in images in otherwise empty paragraphs](#) on page 713

§24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744

21.3.11 Eliminating invisible paragraphs

One way to put “invisible” text into FrameMaker is to make the text white. FrameMaker does not display white text, unless it is against a dark background. You can make **Mif2Go** ignore white text, so that paragraphs containing only white text are removed entirely.

To omit from HTML output paragraphs that contain only white text:

```
[HTMLOptions]
; HideWhiteText removes any white text (standard FrameMaker behavior)
HideWhiteText = Yes
```

If your FrameMaker document contains *any* white text you want to retain, you must set `HideWhiteText=No`, and get rid of the unwanted instances either by assigning their formats the `[HTMLParaStyles]Delete` property (see §21.3.12 [Eliminating unwanted paragraphs](#) on page 652), or by using FrameMaker conditional text, as appropriate.

21.3.12 Eliminating unwanted paragraphs

Suppose your FrameMaker document contains manually inserted page-oriented navigation aids, such as the text “(Continued)” when a procedure breaks across a FrameMaker page boundary. To prevent this text from appearing in HTML output, you can use conditional text, or you can do the following:

1. Use a special paragraph format for all instances of the text in your document.
2. In the configuration file, assign property `Delete` to the paragraph format:


```
[HTMLParaStyles]
; Delete removes the style and all of its content
ParaFmt = Delete
```

The `Delete` format property works whether or not the paragraph actually has content. It omits paragraph content, and any Frames Above/Below, from HTML output, and omits the format from any consideration in mapping to DITA parent elements.

Any paragraph content is still available, and can be used in **Mif2Go** macros; see §28.3.7 [Creating macro variables from paragraph content](#) on page 802.

Note: Applied to an anchor paragraph, `Delete` does not remove any anchored frames or tables that are anchored in the paragraph; `Delete` merely causes the anchor paragraph itself to be omitted from HTML output.

21.4 Mapping character formats

You can specify HTML tags to be used for character formats; for example:

```
[CharTags]
; Document character format name = HTML starting element name(s)
; use strong, em, code, cite, var, or blink, or span for HTML elements
Emphasis=strong
ProgramListing=code
```

Although the format properties listed here are the only valid HTML styles, **Mif2Go** lets you specify any tag, to permit XML markup and CSS `span` class assignments. However, only valid HTML tags have an effect in **Mif2Go** HTML output.

For XHTML, all format names must be lowercase.

Include attributes To add attributes to a character tag, list them after the tag. For example:

```
[CharTags]
Bold = strong type="bold"
```

Everything after the first space is removed for the end tag. To apply an attribute to an individual instance of a character format, insert an attribute marker in the character span; see §29.2.4 [Using attribute markers for HTML or XML](#) on page 835. For this example, you would use a marker of type **CharType** with content `bold` (no quotation marks).

If no tags are specified in `[CharTags]` for a particular character format, by default that format gets a `span` class; see §22.7.3 [Mapping character formats to tags or span classes](#) on page 693.

Suppress tags To eliminate style tags entirely, map the character format to nothing:

```
[CharTags]
CharFmt =
```

See also:

§14.4.2 [Deriving XML tags from format and class names](#) on page 462

§22.7.3 [Mapping character formats to tags or span classes](#) on page 693

21.5 Assigning properties to text formats

You can override some FrameMaker paragraph and character format properties directly, and you can assign additional properties to paragraph and character formats.

In this section:

§21.5.1 [Understanding where to specify format property overrides](#) on page 654

[§21.5.2 Overriding paragraph alignment and size properties](#) on page 656

[§21.5.3 Overriding properties added by typographic elements](#) on page 657

[§21.5.4 Overriding properties specified in font tags](#) on page 657

21.5.1 Understanding where to specify format property overrides

In prior versions of **Mif2Go**, you could specify overrides to both character (inline) formats and paragraph (block) format properties in section [HTMLStyles]. That section is now deprecated, in favor of two new sections:

[HTMLParaStyles] for paragraph overrides

[HTMLCharStyles] for character overrides.

Some former [HTMLStyles] format properties can be assigned either to paragraph formats or to character formats, and so can be used in either of the new sections.

[Table 21-1](#) lists all the properties alphabetically, and shows the sections in which they are valid: **Para** for [HTMLParaStyles] and **Char** for [HTMLCharStyles].

Table 21-1 HTML properties for paragraph and character formats

Format property	Purpose	Para	Char	Ref.
Abbr	Gets value for abbr attribute from [StyleCellAbbr]	X		26.2.2.2
AbbrVal	Make content into abbr for table cell	X		26.2.3
ALink	Uses content for ALink Name property of ALink object	X		7.6.4.2
Alt	Makes content into alt attribute for next 	X		25.2.2
Anum	Includes autonumber in format	X		21.3.3
Axis	Gets value for axis attribute from [StyleCellAxis]	X		26.2.2.2
AxisVal	Makes content into axis for table cell	X		26.2.3
Bold	Encloses text in this format in ...	X	X	21.5.3
CellAttribute	Applies attributes in [StyleCellAttribute] to enclosing cell	X		24.4.6
Center	Centers text between left and right margins	X		21.5.2
CodeAfter	Puts code from [ParaStyleCodeAfter] or [CharStyleCodeAfter] after closing tag	X	X	28.9.3
CodeAfterAnum	Puts code from [AnumCodeAfter] after autonumber	X		28.9.3
CodeBefore	Puts code from [ParaStyleCodeBefore] or [CharStyleCodeBefore] before opening tag	X	X	28.9.3
CodeBeforeAnum	Puts code from [AnumCodeBefore] before autonumber	X		28.9.3
CodeEnd	Puts code from [ParaStyleCodeEnd] or [CharStyleCodeEnd] before closing tag	X	X	28.9.3
CodeReplace	Replaces content with code from [ParaStyleCodeReplace] or [CharStyleCodeReplace]	X	X	28.9.3
CodeStart	Puts code from [ParaStyleCodeStart] or [CharStyleCodeStart] after opening tag	X	X	28.9.3
CodeStore	Stores content in macro variable named in [StyleCodeStore]	X		28.3.7.2
ColGroup	Marks enclosing cell as a header cell that starts a column group	X		26.2.2.2
ColorN	Makes text color N, where N is in the range 1 - 254	X	X	21.5.4
Comment	Makes paragraph a comment, replacing tags	X	X	21.3.6
Config	Makes content act as a configuration override for all outputs	X		33.3
Contents	Includes content in this format in the TOC	X		7.4.3
CSSReplace	Gets code for CSS from [ParaStyleCSS] or [CharStyleCSS]	X	X	22.8.4
Delete	Omits content from text output	X	X	21.3.12

Table 21-1 HTML properties for paragraph and character formats (continued)

Format property	Purpose	Para	Char	Ref.
DListDD	Uses dd instead of dt for items in dl lists	X		21.12.2.1
DropDown	Content is a block to be expanded; bracket with macros	X		7.9.3.1
DropDownBlock	Content is a block to be expanded	X		7.9.3.1
DropDownEnd	Content is last paragraph in expandable block	X		7.9.3.1
DropDownLink	Content is a link to expandable text	X	X	7.9.3.1
DropDownStart	Content is a link, next content is an expandable block	X		7.9.3.1
ExtrDisable	Turns off extract processing	X		18.3.1
ExtrEnable	Turns on extract processing	X		18.3.1
ExtrEnd	Paragraph ends an extract, but is not part of the extract	X		18.3.2.1
ExtrFinish	Paragraph is the last item in an extract	X		18.3.2.1
ExtrStart	Paragraph begins an extract	X		18.3.2.1
Figure	Uses paragraph for anchor tag to ensure wrapping image in <fig>	X		15.7.2
FileName	Uses content to name split files	X		34.8.4.1
FrameAbove	Overrides [HTMLOptions]RemoveFramesAbove=Yes	X		21.3.7
FrameBelow	Overrides [HTMLOptions]RemoveFramesBelow=Yes	X		21.3.7
GlossTerm	Uses content for a glossary term in JavaHelp	X		11.7.2
GlossTitle	Content is a key to hover text		X	13.11
HTMConfig	Makes content act as a configuration override for HTML output	X		33.3
Ital	Italics: encloses text in this format in <i>...</i>	X	X	21.5.3
KeepLink	Retains the first hypertext link in text replaced via [CodeReplace]	X	X	13.8.1.3
Left	Aligns text with left margin	X		21.5.2
LEnd	Non-list format that ends any prior lists	X		21.12.2.1
LFirst	Content in this format starts a list	X		21.12.2.1
LinkClass	Makes content into CSS class attribute value	X		25.3.2
LinkSrc	Puts code from [StyleLinkSrc] in href attribute	X	X	28.9.3
LinkTitle	Makes content into title attribute value	X		25.3.2
ListN	List1 - List12 specify different list styles	X		21.12.2.1
LLevelN	LLevel1 - LLevel30 specify nesting levels	X		21.12.2.1
LNest	Nests in an enclosing list	X		21.12.2.1
Longdesc	Makes content into longdesc attribute value	X		25.2.2
Meta	Makes content a <meta content=...> attribute value	X		18.4.3
NoAnum	Excludes autonumber from non-list formats	X		21.3.3
NoBreak	Changes any Shift+Enter to space instead of for HTML and XHTML	X	X	21.3.8
NoColID	Prevents assignment of id for ColIDs (enabled in [Tables])	X		26.2.2.2
NoColor	Omits for this format	X	X	21.5.4
NoContLink	Suppresses linkage for the corresponding TOC item in HTML Help	X		9.9.5
NoCSS	Omits any CSS entry for this format	X	X	22.8.4
NoFig	Uses paragraph for anchor tag to prevent wrapping image in <fig>	X	X	15.7.2
NoFrameAbove	Overrides [HTMLOptions]RemoveFramesAbove=No	X		21.3.7
NoFrameBelow	Overrides [HTMLOptions]RemoveFramesBelow=No	X		21.3.7
NoHref	Suppresses tags	X	X	13.8.1.5
NoPara	Eliminates only paragraph tags, for use in macros	X	X	21.3.6
NoRef	Forces links to top of page by suppressing part of link after file name	X	X	19.3.1

Table 21-1 HTML properties for paragraph and character formats (continued)

Format property	Purpose	Para	Char	Ref.
NoSize	Omits size attribute from tags for this format	X	X	21.5.4
NoSplit	Prevents format from interfering with SmartSplit	X		18.2.2.4
NoTags	Suppresses attributes and omits any tags between <p> and </p>	X	X	21.3.6
NoWrap	Suppresses \n line breaks and preserves leading spaces	X		21.3.6
Overrides	Allows format properties Bold, Ital, Uline, and Strike	X	X	21.5.3
Plain	Turns off Bold, Ital, Uline, and Strike format properties	X	X	21.5.3
ParaLink	Prevents character spans from affecting a link in the paragraph	X		5.10.2
ParaLinkClass	Links have a class assigned in [StyleParaLinkClass]	X		19.2.2.2
Raw	Suppresses all tags to allow macro inclusions between document elements	X	X	21.3.6
Right	Aligns text with right margin	X		21.5.2
RowAttribute	Attributes in [StyleRowAttribute] are applied to enclosing row	X		24.4.5
RowGroup	Marks enclosing cell as a header cell that starts a row group	X		26.2.2.2
Scope	Gets value for scope attribute from [StyleCellScope]	X		26.2.2.2
SizeN	Size1 - Size7 sets font size attribute to 1 through 7, corresponding to maximum point sizes 8, 10, 14, 20, 28, or 36	X		21.5.2
Span	Causes assignment of ColSpanID or RowSpanID, as enabled in [Tables]	X		26.2.2.2
Split	Starts a new HTML page	X		18.2.1
Strike	Strikethrough: enclose text in this format in <strike>...</strike>	X	X	21.5.3
Summary	Makes content into summary for table tag	X		25.4.3.2
TableBody	Forces containing cell tag to td instead of th	X		26.2.2.4
TableHead	Forces containing cell tag to th instead of td	X		26.2.2.4
TableTitle	Makes content into title attribute for table	X		25.4.3.2
TextFrameIsText	Renders text frames anchored to paragraph as inline text	X		21.3.4
TextStore	Stores content in macro variable named in [StyleTextStore]	X		28.3.7.1
Title	Content in this format becomes HTML page title	X		13.4.5
Trail	Includes content in breadcrumb trail of links	X		20.2.3
Uline	Underline: encloses text in this format in <u>...</u>	X	X	21.5.3
Window	Opens topic in window named in [StyleWindow] for HTML Help	X		9.8.3.1
XMLBreak	Closes tag at Shift+Enter; overrides [HTMLOptions]XMLBreakPara=No	X	X	14.4.5
XMLNoBreak	Changes Shift+Enter to space; overrides [HTMLOptions]XMLBreakPara=Yes	X	X	14.4.5

21.5.2 Overriding paragraph alignment and size properties

To override paragraph alignment and size properties:

```
[HTMLParaStyles]
; Paragraph format = keywords for properties
; Left, Center, Right: alignment properties
; Size1 - Size7 apply those props to head
```

The alignment properties (Left, Center, Right) override the align attribute in the paragraph tag. For the size properties, see §21.7.3 [Mapping font sizes](#) on page 664.

To omit align attributes from *all* paragraph tags:

```
[HTMLOptions]
; AlignAttributes = Yes (default)
; or No (no align attribute in paragraph tags)
```

```
; Default is reversed to No if UseCSS=Yes.
AlignAttributes = No
```

If you use CSS, the default value of `AlignAttributes` is reversed to No; see §22.5 [Understanding how CSS affects other options](#) on page 687.

21.5.3 Overriding properties added by typographic elements

To override properties added by typographic elements:

```
[HTMLParaStyles] or [HTMLCharStyles]
; Format (para or char) = keywords for functions and properties
; Bold, Ital, ULine, and Strike apply those char props to text
; Plain turns all four of those char properties off by default.
```

These are properties added by typographic elements, as opposed to CSS. The use case is for browsers that have poor support for CSS, such as JavaHelp and, to some degree, Eclipse Help. For current popular browsers, you are better off using CSS. However, if you are stuck with a company-mandated CSS and want to tweak something, you can use these properties as overrides.

To eliminate bold, italic, underline, and strikethrough properties (``, `<i>`, `<u>`, and `<strike>` tags) from selected paragraph or character formats:

```
[HTMLParaStyles] or [HTMLCharStyles]
Format = Plain
```

No overrides **Mif2Go** preserves character overrides by default. To eliminate bold, italic, underline, and strikethrough properties (``, `<i>`, `<u>`, and `<strike>` tags) from *all* paragraph and character formats:

```
[HTMLOptions]
; AllowOverrides = Yes (default) or No (ignore untagged char props,
; default for XML and DITA)
AllowOverrides = No
```

Override “no overrides”

You can still allow certain overrides on a format-by-format basis:

```
[HTMLParaStyles] or [HTMLCharStyles]
; Overrides allows text prop overrides for <b>, <i>, <u>,
; and <strike>, even if [HTMLOptions]AllowOverrides is turned off
Format = Overrides
```

See also:

§21.8 [Managing typographic elements for HTML or XML](#) on page 667

21.5.4 Overriding properties specified in font tags

To override paragraph or character properties added as `` attributes:

```
[HTMLParaStyles] or [HTMLCharStyles]
; Color1 - Color254 color text as defined in Frame and [Colors]
; NoColor suppresses use of the <font color=...> in the style.
; NoSize eliminates the size attribute of the font tag, if used.
```

These properties affect the `` tags used within a paragraph or character span for its default style, provided you are including `` tags. Using CSS turns `` tags off by default; see §21.7 [Mapping fonts](#) on page 663 and §22.5 [Understanding how CSS affects other options](#) on page 687.

The `Colornnn` properties use color numbers to assign text colors to paragraph or character formats; see §21.9 [Specifying text colors for HTML](#) on page 669.

21.6 Mapping special characters

Special-character handling is not a strong point of HTML. **Mif2Go** automatically maps characters with ASCII codes 128 through 159, the “high ASCII” characters, to equivalent HTML character entity references. You can specify other character mappings, or even prevent **Mif2Go** from mapping special characters.

In this section:

- §21.6.1 [Understanding how Mif2Go represents characters](#) on page 658
- §21.6.2 [Understanding how Mif2Go treats tabs in HTML/XML](#) on page 658
- §21.6.3 [Understanding Mif2Go support for FrameMaker 8+ Unicode](#) on page 659
- §21.6.4 [Converting Western European accented characters](#) on page 660
- §21.6.5 [Mapping individual special characters](#) on page 660
- §21.6.6 [Mapping characters in a special font](#) on page 662
- §21.6.7 [Avoiding use of special characters in URIs](#) on page 663
- §21.6.8 [Preventing character mapping](#) on page 663

See also:

- §13.4.3 [Specifying character encoding for HTML](#) on page 431
- §13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454
- §14.3.3 [Specifying character encoding for generic XML](#) on page 460

21.6.1 Understanding how Mif2Go represents characters

Mif2Go has two internal ways to represent text: as printable strings, or as single characters. For compactness, **Mif2Go** uses the single-character form for all of the following:

- characters not in the regular printable set, such as curly quotes (straight quotes are in the printable set)
- a printable string that consists of only one character
- a printable character to which a character format has been singly applied.

*Mapped entity
references use
Unicode*

The high ASCII characters, which are not in the printable set, are heavily used in Windows. The ASCII codes for these characters are not valid in Unicode. However, the same glyphs occur in Unicode at other code points, so **Mif2Go** first maps them to their Unicode counterparts. For example, a bullet character in your FrameMaker document becomes numeric entity reference `•` in HTML. The ASCII decimal code for a bullet is 149, whereas the Unicode decimal code for a bullet is 8226. This mapping is applied only to text in the single-character form.

If you specify the following option, **Mif2Go** omits some high ASCII characters and maps others to printable characters:

```
[HTMLOptions]
ValidOnly = Yes
```

See §13.16.2 [Replacing high ASCII characters for W3C validation](#) on page 454.

21.6.2 Understanding how Mif2Go treats tabs in HTML/XML

Because tab characters in text are not meaningful in HTML or XML, by default **Mif2Go** converts each tab in your FrameMaker document to a space, with the following exceptions:

[Tabs in preformatted text or in comments](#)

Tabs in autonumbers

Tabs you replace with code.

HTML and XML coalesce spaces as part of normalization. Several tabs in a row become one space on output, except in preformatted elements or in comments.

*Tabs in
preformatted text
or in comments*

Tabs in the following places become multiple spaces, based on **Mif2Go** line-position calculations that determine where the next “tab stop” should be:

- text in formats mapped to preformatted elements, such as `<pre>` in HTML
- text enclosed in comment tags.

The line-position calculations are accurate only for monospaced fonts. See §21.10 [Configuring preformatted text for HTML/XML](#) on page 670.

*Tabs in
autonumbers*

Mif2Go converts tabs in autonumbers to spaces. However, for paragraph formats assigned the `NoAnum` property, and when `[HTMLOptions]UseAnums=No`, **Mif2Go** removes tabs within or immediately following the autonumbers. See §21.3.3 [Converting paragraph formats with autonumbers](#) on page 648.

*Tabs you replace
with code*

If your document uses tabs for a special purpose, you can replace the tab characters in selected formats with HTML code. The HTML code can be in the form of a **Mif2Go** macro (see §28 [Working with macros](#) on page 787). For example:

```
[StyleTabReplace]
; doc style = HTML code to use instead of tab sequence, can be macro
Flags = <$ReplaceTabWithTD>

[ReplaceTabWithTD]
</td><td>
```

You can assign HTML code to either a character format or a paragraph format. The code is included once for each tab sequence, not for each tab, so if your document has one tab in one paragraph and three in a row in another, they will be treated the same way. Any open inline elements such as `<i>` are closed before the code is inserted, and any new tags for the following text are opened after the code.

21.6.3 Understanding Mif2Go support for FrameMaker 8+ Unicode

FrameMaker (version 8 and later versions) supports only one part of Unicode, the BMP (Basic Multilingual Plane), which is from `U+0000` to `U+FFFF`. FrameMaker does not support the “astral planes”, above `0xFFFF`. FrameMaker does *store* the characters in those planes, using 4-byte UTF-8 encoding, but will not display them, showing just a placeholder (the question mark or a space) instead. However, if you export to HTML using **Mif2Go**, the characters will appear in the output, and anyone with the appropriate font installed can see them in their browser.

Mif2Go provides improved Unicode processing for FrameMaker version 8 and later versions, supporting characters in the “astral planes” above Plane 0 (the Basic Multilingual Plane or BMP), such as the characters in Plane 1 (the Supplementary Multilingual Plane or SMP) and Plane 2 (the Supplementary Ideographic Plane or SIP). UTF-8 can encode characters through Plane 31, if any are ever defined; only Planes 0, 1, 2, 14, 15, and 16 are used in the current specification (Version 5.0). Each plane can include 64K characters. **Mif2Go** produces CJK output using UTF-8, including characters above the BMP, such as Chinese Extension B. Viewing such characters requires a font that shows them; but even if such a font is not present on your system, **Mif2Go** produces the correct characters.

Mif2Go does not support non-Unicode double-byte character sets, except for Asian and Cyrillic code pages used by HTML Help; see §9.13 [Generating HTML Help in non-](#)

[Western languages](#) on page 331 and §13.4.3 [Specifying character encoding for HTML](#) on page 431.

See also:

§22.7.6 Assigning CSS classes based on Unicode character ranges on page 694

§38.2.4 Saving FrameMaker 8 files as FrameMaker 8 MIF on page 1008

21.6.4 Converting Western European accented characters

Mit freundlichen Grüßen. **Mif2Go** converts Western European languages to HTML; your text and tags should appear as usual, except that CSS class names cannot contain accented characters. For class names, where possible **Mif2Go** replaces an accented character with the corresponding non-accented character; see §22.7.1 [Understanding CSS class name restrictions](#) on page 691.

21.6.5 Mapping individual special characters

To force a mapping different from the **Mif2Go** mapping of a particular character, or to map any arbitrary Unicode character (for example):

```
[CharConvert]
; Unicode char num = HTML numeric value or string replacement
; nonbreaking hyphen is decimal 8209, becomes entity &#150;
8209 = 150
; em space is x2003, becomes three nonbreaking spaces
x2003 = &nbsp;&nbsp;&nbsp;
```

Character to
replace

To the left of the equals sign, specify any of the following for the character you want to replace:

- the decimal ASCII character code; you can find these codes in `FrameMaker Character_Sets.pdf`, in the `OnlineManuals` directory
- the decimal Unicode character number
- x followed by the hexadecimal code for the character
- u+ or U+ followed by the hexadecimal code for the character
- the character itself, if it is one of the following:
 - a character in the printable set other than the asterisk (*) or question mark (?), both of which **Mif2Go** treats as wildcards unless you disable this feature; see §5.1.7 [Specifying how to treat cases, spaces, and wildcards](#) on page 113
 - a high ASCII character (decimal code 128 through 159).

The easiest way to specify a character to be replaced (except asterisk or question mark) is to copy and paste the character from your FrameMaker document into section [CharConvert]. This works for most symbols, but not for variant spaces, which turn into plain spaces, nor for hard or soft hyphens, which turn into plain hyphens. For these characters, and for a few others, [Table 21-2](#) shows the Unicode or other hexadecimal (and in some cases, decimal) value you can specify to the left of the equals sign.

Table 21-2 Special characters to replace for HTML/XML output

Category	Character to replace	Unicode/Hex	ASCII decimal
Quote marks	Low single quote	x201A	130
	Left single quote	x2018	145
	Right single quote	x2019	146
	Low double quote	x201E	132
	Left double quote	x201C	147
	Right double quote	x201D	148
Spaces	Hard space	x00A0	160
	En space	x2002	---
	Em space	x2003	---
	Numeric (figure) space	x2007	---
	Thin space	x2009	---
Dashes	En dash	x2013	150
	Em dash	x2014	151
Hyphens	Discretionary hyphen	x00AD	173
	Nonbreaking hyphen	x2011	---
Wildcards	Asterisk	x002A	042
	Question mark	x003F	063
Miscellaneous	Bullet	x2022	149
	Fraction bar	x2044	---
	Paragraph symbol	x00B6	182
	Section symbol	x00A7	167

Replacement character

To the right of the equals sign, specify any of the following:

- the decimal ASCII character code for the replacement character
- x followed by the hexadecimal code for the replacement character
- a string, which can include HTML code and **Mif2Go** macro references.

When you supply a string rather than a character code, **Mif2Go** expands any macros referenced, but includes the rest of the string in the output *as is*. Therefore you must escape any literal characters such as < by providing an entity reference instead; in this case, <.

Examples

To map the bullet to a middle dot:

```
[CharConvert]
149 = 183
```

To map the bullet to a bold middle dot:

```
[CharConvert]
149 = <b>&#183;</b>
```

To map the bullet to an image:

```
[CharConvert]
149 = 
```

To map the ohm symbol from Unicode to the Symbol font for HTML Help:

```
[CharConvert]
U+2126 = <span class="Symbol">W</span>
```

and add the class to your CSS:

```
[CSSEndMacro]
.Symbol {font-family: Symbol; }
```

In code-page encoding, as for HTML Help output, the only valid solution for handling out-of-range characters is to use a font that has the desired glyph within the code page. In this case, the glyph for ohm is in Symbol, which will work in all single-byte code pages (but not in Asian code pages, where an Asian symbol font is needed instead).

To prevent **Mif2Go** from mapping the curly right single quote to its corresponding HTML entity, and replace it instead with a straight apostrophe (which is in the printable set):

```
[CharConvert]
146 = '
```

*Use only to map
non-printable
characters*

Although you can specify any decimal integer to the left of the equals sign, this mapping option is intended only for characters that are not in the regular printable set. Using [CharConvert] to map a character in the printable set can result in surprises. You can try mapping other integers, but the odds are poor for values not in the range 128 through 255. There are a few exceptions. For example, **Mif2Go** automatically converts a solidus to a forward slash, which is in the printable set. You can prevent this conversion by mapping the solidus to itself, specifying the Unicode value to the left of the equals sign and again on the right, as a numeric entity reference:

```
[CharConvert]
8260 = &#8260;
```

See §21.6.1 [Understanding how Mif2Go represents characters](#) on page 658.

Font is ignored

Character mapping via [CharConvert] takes no notice of font. For example, if you map a character that appears as a check mark in WingDings 2 to the Unicode radical sign, any instance of capital “P” in your document that occurs by itself (away from other letters, or with a character format applied to it alone) will appear as a radical sign in HTML output. This is because in the WingDings 2 character set, the check mark has ASCII decimal code 80, the same code as a capital “P” in the standard character set:

```
[CharConvert]
; WingDings 2 check marks (and individual text Ps) become radicals:
80 = 8730
```

To get around this problem, see §21.6.6 [Mapping characters in a special font](#) on page 662.

21.6.6 Mapping characters in a special font

Characters in special fonts such as Wingdings or Webdings might not be rendered correctly by non-Microsoft browsers; see §21.7.7 [Accommodating browser font-rendering differences](#) on page 666. However, you can direct **Mif2Go** to replace a character in any font with its Unicode equivalent, or with an image, or with any HTML code, by providing a macro section for that font in your configuration file.

The following Web site suggests mappings from Symbol and Wingdings characters to the closest Unicode code points:

<http://www.alanwood.net>

With the kind permission of Alan Wood, we have incorporated these mappings as defaults for all **Mif2Go** HTML conversions. The defaults should handle all Symbol characters (except for one, the radical extender) and the majority of Wingdings characters. You can override the defaults, or set mappings for any characters that have no equivalent Unicode representation.

To specify a macro section for Wingdings (for example):

```
[MacroFonts]
; Frame font name = section to use for mapping chars in that font
Wingdings = WingChars
```

In the macro section, you can represent an individual character as itself, as its decimal value, or (prefixed with `x`) as its hexadecimal value; and you can map it to its Unicode equivalent, to a string, to HTML code, or to a **Mif2Go** macro. The rules are the same as for mapping individual characters in section [CharConvert]; see §21.6.5 [Mapping individual special characters](#) on page 660.

For example:

```
[WingChars]
; char n maps to a square bullet, char p to a graphic:
n = x25a0
p = 
```

To produce the Unicode equivalent of **ž** and **ž** (Z caron and z caron) listed in [Table 13-6](#) on page 454, for example:

```
[MacroFonts]
Courier New = ZCaron

[ZCaron]
142 = U+017D
158 = U+017E
```

To map characters that are not in the printable set, see §21.6.5 [Mapping individual special characters](#) on page 660.

21.6.7 Avoiding use of special characters in URIs

URI (Uniform Resource Identifier) encoding rules are different from HTML and XML encoding rules. There is no way to URI-encode characters that have a decimal value greater than 255; you get only eight bits for each character. **Mif2Go** does not know that a particular attribute value or text string is intended to become part of a URI, and by default converts any characters outside this range to numeric entities. Therefore, avoid special characters such as trademarks and registration marks in any Internet or email addresses in your document.

21.6.8 Preventing character mapping

You can prevent **Mif2Go** from mapping high ASCII characters to entity references:

```
[HTMLOptions]
Encoding = None
```

However, this option does not produce valid HTML; see §13.16 [Passing W3C validation tests](#) on page 453.

See also:

§13.4.3.3 [Specifying encoding for double-byte characters](#) on page 432

21.7 Mapping fonts

Try to keep your font usage in HTML very simple, using only fonts that you are certain your readers have on their systems. A browser's substitution for an unavailable font can be quite ugly.

In this section:

§21.7.1 [Specifying a default font and size](#) on page 664

§21.7.2 [Remapping fonts](#) on page 664

§21.7.3 [Mapping font sizes](#) on page 664

§21.7.4 [Including or excluding font tags](#) on page 665

§21.7.5 [Managing font tags for symbol fonts](#) on page 666

§21.7.6 [Excluding face and size attributes from font tags](#) on page 666

§21.7.7 [Accommodating browser font-rendering differences](#) on page 666

21.7.1 Specifying a default font and size

You can specify the default font and size to use; for example:

```
[Base]
; Font name and size put out at start of file, default none
Font=Times
Size=3
```

Not all browsers respect this setting. If you use CSS, the default value of Basefont is reversed to No; see §22.5 [Understanding how CSS affects other options](#) on page 687.

You can suppress the <basefont> tag entirely, or omit the face attribute while retaining size:

```
[HTMLOptions]
; Basefont = Yes (default) or No (no <basefont> tag put out)
; Default is reversed to No if UseCSS=Yes.
Basefont = No
; UseFontSize = Yes (default, allow size attrib in font tags) or No;
; reversed for Eclipse Help and JavaHelp
UseFontSize = No
; UseFontFace = Yes (default, allow face attrib in font tags) or No
UseFontFace = No
```

For Eclipse Help and JavaHelp, the default value of UseFontSize is reversed to No.

If you do not use CSS, and you are not using tags either, you will get whatever fonts a browser specifies as defaults.

21.7.2 Remapping fonts

You can remap the fonts used in your FrameMaker document; for example:

```
[Fonts]
; Document font name = HTML font name (comma-delimited list allowed)
Helvetica = Arial,MSSansSerif
Myria* = Arial
NewCenturySchlbk = Times
Century Schoolbook = Times
```

You can use wildcards to change all font names that share a base name. This is helpful with multiple-master fonts, where you might have many very long names.

See also:

§22.8.1 [Assigning a CSS generic font family](#) on page 698

21.7.3 Mapping font sizes

For CSS, Mif2Go shows the font size just as it is in FrameMaker. For HTML itself, Mif2Go must convert point size to an HTML size number, 2 through 7. To modify the way Mif2Go maps the size, you can replace all or part of this table:

```
[FontSizes]
; HTML font size = pt size it starts with for default usage
; for example, if 3=10 and 4=14, 12pt type becomes size=3
; computed size is overridden by [HTMLParaStyles] or [HTMLCharStyles]
; SizeN setting
2 = 8
3 = 10
4 = 14
5 = 20
6 = 28
7 = 36
```

The number on the right side of the equals sign is the largest point size to be rendered as the HTML size number on the left. In the example above, 8-pt and smaller is size 2, 9-pt and 10-pt are size 3, and so on. To make 10-pt text appear as size 4, you would lower the limit for size 3; for example, 3=9. If the size 4 font is too large and heavy, try changing 4=14 to 4=15. That makes `` for 14-point text, by raising the start of the size 4 range to 15-point text.

The `[FontSizes]` settings determine how **Mif2Go** converts from points to HTML size numbers, regardless of any other settings. Also see §21.5.2 [Overriding paragraph alignment and size properties](#) on page 656.

To change CSS entries from points to other size units, see §22.8.3 [Specifying CSS size values and units of measurement](#) on page 699.

21.7.4 Including or excluding font tags

Older versions of Internet Explorer contain a defect in how `` tags are handled. If your HTML output includes a `` tag, and the specified font does not include the glyph, Internet Explorer changes the glyph to another that does occur in that font, and does a poor job of selection. Firefox simply ignores the `` tag and shows the correct character, in compliance with the W3 specification.

By default, when you use CSS, **Mif2Go** does not include `` tags in HTML output, except for symbols in autonumbers; see §21.7.5 [Managing font tags for symbol fonts](#) on page 666. However, you might need additional `` tags in some circumstances. For example:

- If you use an OpenType or TrueType font, some browsers require `` tags to correctly display content in these fonts; see §21.7.7 [Accommodating browser font-rendering differences](#) on page 666.
- If you are producing JavaHelp or Oracle Help, you might need `` tags to get around viewer deficiencies; see §11.3.9 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 384.

To turn on `` tags in HTML output:

```
[HTMLOptions]
; NoFonts = Yes (default, prohibit <font ...> tags except for symbol
; fonts) or No (use <font...> tags, default if UseCSS=No)
NoFonts = No
```

If you turn off CSS, **Mif2Go** turns on `` tags by default; see §22.5 [Understanding how CSS affects other options](#) on page 687. If you do use CSS, you can create `` tags instead, with a single setting for each character format; see §22.7.3 [Mapping character formats to tags or span classes](#) on page 693.

If you do not use CSS, and you are not using `` tags either, you will get whatever fonts a browser specifies as defaults.

21.7.5 Managing font tags for symbol fonts

By default, **Mif2Go** inserts inner `` tags when the source character format specifies the Symbol font, even when `NoFonts=Yes`. This is done to ensure that bullets from the Symbol font come out right. However, **Mif2Go** suppresses `` tags for characters in the Symbol font (or in any of its brothers, such as Zapf Dingbats, Webdings, or Wingdings) that occur *outside of* autonumbers.

To preserve `` tags when the source character format specifies a symbol font (even when you have explicitly set `NoFonts=Yes`):

```
[HTMLOptions]
; NoSymbolFont = Yes (default, prohibit <font...> tags even for symbol
; fonts), or No (use <font...> tags for symbol fonts)
NoSymbolFont = No
```

When `NoSymbolFont=Yes`, **Mif2Go** maps non-autonumber symbols in your document to appropriate Unicode numeric character entities; see §21.6.6 [Mapping characters in a special font](#) on page 662.

If you set `NoSymbolFont=No`, one alternative is to keep the `` tags, but omit the face attribute; see §21.7.6 [Excluding face and size attributes from font tags](#) on page 666.

21.7.6 Excluding face and size attributes from font tags

To suppress the face attribute in `` tags:

```
[HTMLOptions]
; UseFontFace = Yes (default, allow face attribute in <font>) or No
UseFontFace = No
```

The default value of `UseFontFace` is Yes, except for JavaHelp and Eclipse Help; for those output types, the default is No.

Allowing `` tags for size but not for face avoids interfering with CSS specifications. For example, omitting the face attribute is required for W3C validation of HTML 3.2 for JavaHelp; see §11.3.9 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 384.

On the other hand, if you use non-Unicode-compliant fonts such as Webdings and Wingdings, the only way to get certain non-Microsoft browsers to render characters in those fonts is to use the face attribute; see §21.7.7 [Accommodating browser font-rendering differences](#) on page 666.

To suppress the size attribute in `` tags:

```
[HTMLOptions]
; UseFontSize = Yes (default, allow size attribute in <font>) or No
UseFontSize = No
```

21.7.7 Accommodating browser font-rendering differences

Some browsers (Opera and Safari, for example) do not support OpenType and TrueType fonts. Mozilla browsers (Firefox, for example) support these fonts only when you use `` tags. For example, you cannot get Firefox to render a character in a special font such as Webdings or Wingdings unless you enclose the character (using its standard ASCII equivalent) in a `` tag with the face attribute. For example, you would need the following code to make Firefox display a Wingdings square bullet:

```
<font face="wingdings">n</font>
```

You could direct **Mif2Go** to use `` tags:


```
[HTMLOptions]
NoFonts = No
UseFontFace = Yes
```

However, this workaround is not effective for Opera or Safari, and might not be reliable for any non-Microsoft browser. A better solution is to map any special characters to their Unicode counterparts; see:

§21.6.5 [Mapping individual special characters](#) on page 660

§21.6.6 [Mapping characters in a special font](#) on page 662.

21.8 Managing typographic elements for HTML or XML

By default, for HTML output **Mif2Go** provides typographic elements for FrameMaker paragraph or character formats that specify bold, italic, underline, subscript, or superscript as part of the format. For example, for every paragraph format whose definition includes bold formatting, by default HTML output includes `` elements as well as the code for the paragraph tag.

In this section:

§21.8.1 [Deciding whether to suppress typographic elements](#) on page 667

§21.8.2 [Choosing how to treat typographic elements](#) on page 667

21.8.1 Deciding whether to suppress typographic elements

You might want to suppress some or all typographic elements for either of the following reasons:

- To ensure that output is free of direct formatting, because:
 - you are producing XML output, or
 - your output uses CSS.
- To make overrides show up in the output, so you can insert semantic tags or subsequently provide better formatting in FrameMaker.

For XML output, including DITA XML and DocBook XML, the default is to suppress all typographic elements.

21.8.2 Choosing how to treat typographic elements

To specify how typographic elements should be treated:

```
[Typographics]
; UseTypographicElements = Yes (HTML default) or No (XML default,
; suppress b, i, u, tt, sub, and sup even when specified in a format)
; UseFormatTypographics = Yes (default, use b, i, u, strike, sub
; and sup when set in paragraph or character formats), or No
; (suppress in both para and char formats)
; UseParagraphTypographics = Yes (default, use above when set in
; paragraph formats), or No (suppress in para formats)
; UseCharacterTypographics = Yes (default, use above when set in
; character formats), or No (suppress in char formats)
; UseTypographicStyles = No (default) or Yes (use tags below if set)
; typographic tag (b, i, u, strike, sub, sup) = tag to use instead,
; possibly followed by attributes. Replaces overrides if used
; while [HTMLOptions]AllowOverrides=Yes, UseTypographicElements=Yes,
; and UseFormatTypographics=No.
; Both "over" for overline and "chbar" for change bar can be used
; as pseudotags here.
```

```
[HtmlOptions]
; AllowOverrides = Yes (default) or No (ignore untagged char props,
; default for XML and DITA)
```

You can choose to:

- [Suppress all typographics](#)
- [Suppress typographics used as overrides](#)
- [Suppress typographics in formats](#)
- [Suppress typographics only in paragraph formats](#)
- [Suppress typographics only in character formats](#)
- [Replace typographics with other tags.](#)

*Suppress all
typographics*

To eliminate all typographic elements, both those used as overrides in FrameMaker and those that are intrinsic to a FrameMaker paragraph or character format:

```
[Typographics]
UseTypographicElements = No
```

When UseTypographicElements=No, all settings of character properties are eliminated, including font size, font color, and font name in addition to bold, italic, underline, strike, subscript, and superscript; regardless of whether those properties are intrinsic to a FrameMaker character or paragraph format or were applied as an override. This setting, UseTypographicElements=No, cannot be overridden by any other settings in section [Typographics], nor by [HtmlOptions]AllowOverrides. This is the appropriate value for DITA XML output; see §15.4.4 [Mapping character formats to DITA inline elements](#) on page 492.

*Suppress
typographics
used as overrides*

To suppress only those typographic elements used as overrides, but keep typographic elements intrinsic to formats:

```
[Typographics]
UseTypographicElements = Yes
UseFormatTypographics = Yes

[HtmlOptions]
AllowOverrides = No
```

When AllowOverrides=No, bold, italic, underline, strike, subscript, and superscript are eliminated when they are applied as overrides in FrameMaker; see §21.5 [Assigning properties to text formats](#) on page 653.

*Suppress
typographics in
formats*

To eliminate typographic elements that are intrinsic to formats, but keep overrides:

```
[Typographics]
UseTypographicElements = Yes
UseFormatTypographics = No

[HtmlOptions]
AllowOverrides = Yes
```

When UseFormatTypographics=No, intrinsic format properties eliminated include bold, italic, underline, strike, subscript, and superscript; however, font size, font color, and font name are retained in the output.

*Suppress
typographics only
in paragraph
formats*

To eliminate typographic elements intrinsic to paragraph formats, but keep those intrinsic to character formats:

```
[Typographics]
UseTypographicElements = Yes
UseParagraphTypographics = No
UseCharacterTypographics = Yes
```

*Suppress
typographics only
in character
formats*

When `UseParagraphTypographics=No`, format properties intrinsic to paragraph formats are eliminated; these include bold, italic, underline, strike, subscript, and superscript. Font size, font color, and font name are retained in the output.

To eliminate overrides and typographic elements intrinsic to character formats, but keep those intrinsic to paragraph formats:

```
[Typographics]
UseTypographicElements = Yes
UseParagraphTypographics = Yes
UseCharacterTypographics = No

[HtmlOptions]
AllowOverrides = No
```

When `UseCharacterTypographics=No`, format properties intrinsic to character formats are eliminated; these include bold, italic, underline, strike, subscript, and superscript. Font size, font color, and font name are retained in the output.

*Replace
typographics with
other tags*

To specify tags to use for individual typographic elements:

```
[Typographics]
UseTypographicElements = Yes
UseTypographicStyles = Yes
typographic = tag
```

When `UseTypographicStyles=Yes`, you can specify other tags to use in place of typographic elements. The typographic elements you can replace are `b`, `i`, `u`, `strike`, `sub`, and `sup`. You can also replace “pseudo tags” over for overline, and `chbar` for change bar. You can specify attributes as well. For example:

```
[Typographics]
UseTypographicStyles = Yes
i = emphasis
b = emphasis role="bold"
```

If `UseFormatTypographics=Yes`, the tags you specify replace any named typographics intrinsic to formats.

If `[HtmlOptions]AllowOverrides=Yes`, the tags you specify replace any named typographics used as overrides.

21.9 Specifying text colors for HTML

You can specify colors for both character and paragraph formats. Text color is set in CSS; and also in `` tags, if you leave them enabled; see §21.7.4 [Including or excluding font tags](#) on page 665.

To use colors different from those in your FrameMaker document:

1. Identify the color you want (or define a new color) *by number* (in the range 9-254), and assign to it a hexadecimal color value:

```
[Colors]
nnn = fffffff
```

See §13.7.2 [Mapping FrameMaker colors to new values](#) on page 439.

2. Assign the color, *by number* prefixed with the word `Color`, to the paragraph or character format:

```
[HTMLParaStyles] or [HTMLCharStyles]
; Color1 - Color254 color text as defined in Frame and [Colors]
; NoColor suppresses use of the <font color=...> in the style.
Fmtname = Colornnn
```

For example:

```
[Colors]
; Major headings should be blue:
102 = 0000ff
; Cautionary notes should be red:
99 = ff0033

[HTMLParaStyles]
Heading1 = Color102
Caution = Color99
Sidetip = NoColor
```

21.10 Configuring preformatted text for HTML/XML

Paragraphs to which you have assigned the `pre` format property (for “preformatted” text) become blocks enclosed in `<pre>` tags in HTML. Browsers and other HTML viewers treat text within `<pre>` tags differently from other text. For example, long lines do not wrap when you narrow the viewer window, and whitespace is preserved.

In this section:

[§21.10.1 Eliminating line wraps in preformatted text](#) on page 670

[§21.10.2 Replacing tabs with spaces in preformatted text](#) on page 671.

21.10.1 Eliminating line wraps in preformatted text

For preformatted text in HTML and XML, by default **Mif2Go** retains line breaks exactly as they appear in your FrameMaker document. This applies to all of the following:

- FrameMaker line wraps
- Shift+Enter forced returns
- Paragraph breaks.

To omit only FrameMaker line wraps (typesetting “soft returns”) and show wrapped lines full length in HTML or XHTML `<pre>` elements and XML preformatted elements:

```
[HTMLOptions]
; IgnoreWrap = No (default, \n where wrap occurs) or Yes
IgnoreWrap=Yes
```

When `IgnoreWrap=Yes`, line wraps introduced by FrameMaker are ignored for any `pre` element; see [§21.3.1 Assigning HTML tags and attributes to paragraph formats](#) on page 646.

When `IgnoreWrap=No`, line wraps introduced by FrameMaker in paragraphs mapped to `pre` elements become line breaks in HTML output.

To omit *all* line breaks from text mapped to `pre` elements;

```
[HTMLOptions]
; UnwrapPRE = No (default) or Yes (ignore line breaks in PRE)
UnwrapPRE = Yes
```

When `UnwrapPRE=Yes`, **Mif2Go** ignores all line breaks in text mapped to preformatted elements: those caused by FrameMaker line wraps, those caused by FrameMaker **Shift+Enter** forced returns (typesetting “hard returns”), and those caused by paragraph breaks. `UnwrapPRE` is effective only within `<pre>` elements in HTML and XHTML, and within preformatted elements in XML.

To preserve leading spaces in preformatted text, also assign the following format property to the paragraph format:

```
[HTMLParaStyles]
ParaFmt = NoWrap
```

See also:

§13.6.4 [Suppressing line breaks in HTML and XML output](#) on page 437

§14.4.5 [Configuring forced returns for XML](#) on page 465

§21.3.8 [Deciding how to treat forced returns](#) on page 651

21.10.2 Replacing tabs with spaces in preformatted text

For HTML preformatted text, **Mif2Go** converts tabs to spaces. To specify how many spaces to use per tab inch:

```
[HTMLOptions]
; TabCharsPerInch = count of spaces to use in PRE for 1" of tabbing
; the default, 16, makes 1/4" tabs into 4 chars, and 1/2" tabs into 8
TabCharsPerInch = 16
```

The default works well with 9-pt Courier New, for displaying code sections. In FrameMaker, set tabs at the interval specified to get the same appearance. You can skip unused tab stops. See §21.6.2 [Understanding how Mif2Go treats tabs in HTML/XML](#) on page 658.

21.11 Converting footnotes to HTML or XML

Mif2Go reads FrameMaker document settings for footnote properties (including numbering properties) and sets footnotes for HTML or XML output accordingly. If you are not using CSS (see §22.1 [Deciding whether to use CSS](#) on page 681), **Mif2Go** sets the type attribute of the used for the footnotes. If the document has a custom footnote type, **Mif2Go** uses a <div> with the class instead of , and a <p> instead of , and writes out the symbols.

In this section:

§21.11.1 [Configuring and placing footnotes](#) on page 671

§21.11.2 [Eliminating links to jump footnotes](#) on page 672

§21.11.3 [Using list tags or <div> and <p> tags for jump footnotes](#) on page 672

§21.11.4 [Formatting jump footnote text with macros](#) on page 673

See also:

§22.7.5 [Assigning CSS classes to text and table footnotes](#) on page 694

§24.5.2 [Configuring and positioning table titles](#) on page 747

21.11.1 Configuring and placing footnotes

Mif2Go provides the following options for placement of footnotes from your FrameMaker document:

- embed footnotes in text, [between brackets]
- embed footnotes in text, enclosed in tags
- place footnotes at the end of the output file, after a separator
- omit footnotes entirely.

For table footnotes, see §24.5.4 [Positioning table footnotes](#) on page 748.

To specify placement of footnotes:

```
[HTMLOptions]
; Footnotes = Jump (HTML default, at end), Embed (between []),
;   Inline (XML default), or None
Footnotes = Jump
; FootnoteSeparator is used at end of doc before Jump footnotes
FootnoteSeparator = <br /><br /><hr />
```

Values for Footnotes have the following effects:

- | | |
|--------|--|
| Jump | All footnotes referenced in a file appear at the end of the file. If you are splitting files (see §18 Splitting and extracting files on page 585), the footnotes for each split file appear at the end of that file. Footnote text follows a separator that you can specify by providing a value for FootnoteSeparator. The default value is

<hr />. |
| Embed | Each footnote appears where it is referenced in text, enclosed in square brackets [<i>footnote text</i>], replacing the reference. |
| Inline | Each footnote appears where it is referenced in text, enclosed in tags, replacing the reference. This is the default for XML, DITA, and DocBook. |
| None | Footnote reference and text are both omitted from output. |

To specify configuration of footnotes and footnote links when Footnote=Inline:

```
[HTMLOptions]
; FootInlineTag = tag for beginning and ending inline footnotes
FootInlineTag=footnote
; FootInlineParaTag = tag for beginning and ending inline footnote
;   paras
FootInlineParaTag=para
; FootInlineIDPrefix = start of ID attr for inline footnotes; rest
;   is sequential number starting with 1 at start of file.
FootInlineIDPrefix=foot
; UseFootXrefTag = No (HTML default) or Yes (XML default)
UseFootXrefTag=No
; FootInlineRefTag = tag for xrefs to inline footnotes, uses linkend
;   for href attribute, for DocBook
FootInlineXrefTag=footnoteref
```

21.11.2 Eliminating links to jump footnotes

By default, **Mif2Go** creates a link for each reference to a Jump footnote (see §21.11.1 [Configuring and placing footnotes](#) on page 671).

To eliminate links to footnotes:

```
[HTMLOptions]
; NoFootnoteLinks = No (default) or Yes (eliminate links to footnotes)
NoFootnoteLinks = Yes
```

When NoFootnoteLinks=Yes and Footnotes=Jump (see §21.11.1 [Configuring and placing footnotes](#) on page 671), footnotes appear where and how specified, but references to them do not contain active links.

21.11.3 Using list tags or <div> and <p> tags for jump footnotes

By default, **Mif2Go** uses list tags for Jump footnotes in both text and tables (see §21.11.1 [Configuring and placing footnotes](#) on page 671). For numbered footnotes, **Mif2Go** supplies Arabic numerals, even if your FrameMaker document uses Roman numerals. For alphabetic footnotes, if the quantity of footnotes per page exceeds the length of the alphabet, **Mif2Go** repeats the sequence.

To use <div> and <p> for footnotes instead of list tags:

```
[HTMLOptions]
; UseFootnoteLists = Yes (default, use <ol> and <li> for footnotes
;   in text, except for those using symbols),
;   or No (always use <div> and <p>).
UseFootnoteLists = No
; UseTbFootnoteLists = Yes (default, use <ol> and <li> for footnotes
;   in tables, except for those using symbols),
;   or No (always use <div> and <p>).
UseTbFootnoteLists = No
```

Using <div> and <p> is likely to give better cross-browser consistency; we advise avoiding HTML list tags whenever possible.

See also §22.7.5 [Assigning CSS classes to text and table footnotes](#) on page 694.

21.11.4 Formatting jump footnote text with macros

By default, **Mif2Go** removes any paragraph start/end coding within a footnote. However, for Jump footnotes (see §21.11.1 [Configuring and placing footnotes](#) on page 671) you can provide HTML formatting by specifying macros to precede and follow each footnote.

To surround footnotes with HTML code:

```
[HtmlOptions]
; FootnoteStartCode macro is used after <a name=...>
;   of each Jump footnote
;FootnoteStartCode =
; FootnoteEndCode macro is used at end of each Jump footnote
;FootnoteEndCode =
```

For example, if some footnotes include bulleted lists, you could assign starting and ending macro code to the paragraph formats you use for list items in footnotes. Suppose you use formats `FootBullet1` and `FootBullet2` (for bullet items indented within other bullet items). You could specify the following settings, macros, and macro variables:

```
[HTMLOptions]
FootnoteEndCode = <$FootEnd>

[HtmlParaStyles]
FootBullet1 = CodeStart NoAnum
FootBullet2 = CodeStart NoAnum

[ParaStyleCodeStart]
FootBullet1 = <$FootBullStart1>
FootBullet2 = <$FootBullStart2>

[FootBullStart1]
<$_if ($$F2Started)></ul>\n<$$F2Started=0><$_endif>\
<$_if not ($$F1Started)><ul type="disc">\n<$$F1Started=1><$_endif>\
<li>\

[FootBullStart2]
<$_if not ($$F2Started)><ul type="circle">\n<$$F2Started=1><$_endif>\
<li>\

[FootEnd]
<$_if ($$F2Started)></ul>\n<$$F2Started=0><$_endif>\
<$_if ($$F1Started)></ul>\n<$$F1Started=0><$_endif>\

[MacroVariables]
; Put any macro definition sections before this section.
F1Started = 0
F2Started = 0
```

This macro code provides the proper and coding for the bulleted paragraphs.

Note: In HTML the convention is not to use ``, because this closing tag can create extra unwanted spacing in some browsers. However, `` is required in XHTML and XML.

If you code numbered footnotes as `` items, they should be in an `` block, which provides the numbering. Numbering restarts at 1 for each HTML page. See §21.12.2

[Converting list formats to HTML list styles](#) on page 675.

21.12 Converting list formats to HTML

Lists, especially nested lists, are a challenge to format correctly for HTML output.

In this section:

§21.12.1 [Understanding the problem with HTML lists](#) on page 674

§21.12.2 [Converting list formats to HTML list styles](#) on page 675

§21.12.3 [Indenting list items](#) on page 678

§21.12.4 [Converting list formats to HTML/XML paragraphs](#) on page 679

See also:

§21.3.3 [Converting paragraph formats with autonumbers](#) on page 648

§34.7 [Converting autonumbers for database systems](#) on page 944

21.12.1 Understanding the problem with HTML lists

You might have already discovered that no matter how you map your numbered lists, they do not render correctly in one browser or another. This problem is the result of a difference in how browsers indent list items. The situation is described in Eric Meyer's CSS book for O'Reilly, 3rd Ed., pp. 377-378. Basically, you can indent with either margin or padding. So Internet Explorer and Opera use this:

```
ul, ol {margin-left: 40px; }
```

Firefox and other Gecko browsers use this:

```
ul, ol {padding-left: 40px; }
```

Both methods comply with standards, but they create a compatibility issue. The fix is to override one or the other in your own CSS, depending on how you prefer to indent your own list items. If you use padding, add (for example):

```
ul, ol {margin-left: 0; padding-left: 1em; }
```

If you use margins, add:

```
ul, ol {margin-left: 1em; padding-left: 0; }
```

Mif2Go sets both margin and padding:

```
ul.Bulleted1 {
  margin-left: 18pt;
  padding-left: 12pt;
  list-style: disc;
}

ol.Numbered1 {
  margin-left: 18pt;
  padding-left: 12pt;
  list-style: decimal;
}
```

Mif2Go uses separate rules for `ol` and `ul` because some viewers (notably the JavaHelp viewer) do not follow CSS cascading rules correctly.

21.12.2 Converting list formats to HTML list styles

HTML list tags are far more restrictive than straight CSS. However, if the following are true, you should be able to successfully convert FrameMaker list formats to HTML list styles:

- Either the list formats in your document do not have complex autonumbers, or you can allow those that do to become <p> items in the output instead.
- The list formats in your FrameMaker document indicate when a new list or sublist begins. **Mif2Go** tries to emulate FrameMaker numbering using HTML list tags, but needs guidance from the FrameMaker formats you use to know where nested lists begin, continue, and end.

In most cases browsers will reformat the list styles unaided, and they will come out as you expect. However, see §21.12.3 [Indenting list items](#) on page 678 for ways you might have to modify CSS properties to line up indents.

Note: Unless you specify XHTML as the output type, **Mif2Go** does not generate closing tags, because browsers tend to space poorly when is present.

In this section:

- §21.12.2.1 [Specifying HTML list styles \(deprecated\)](#) on page 675
- §21.12.2.2 [Converting lists with multiple paragraph formats](#) on page 676
- §21.12.2.3 [Converting nested lists](#) on page 676
- §21.12.2.4 [Converting dictionary lists](#) on page 677
- §21.12.2.5 [Including FrameMaker autonumbers in lists](#) on page 677
- §21.12.2.6 [Omitting CSS class attributes from list entries](#) on page 678
- §21.12.2.7 [Including or excluding the type list attribute](#) on page 678

21.12.2.1 Specifying HTML list styles (deprecated)

To specify HTML list styles explicitly:

```
[HTMLParaStyles]
; doc format (para or char) = keywords for functions and properties
; List1 - List12 specify different list styles:
;   1-5 = OL, ordered list types 1, i, I, a, and A
;   6-8 = UL, unordered list types disc, circle, and square
;   The rest vary from one browser to another; Opera shows as:
;   9 = DIR, nonindented list, no bullets or numbers
;   10 = MENU, bulleted and indented like 6
;   11 = DL, dictionary list, indented, no bullets
;   12 = DL COMPACT, like 11 but with less spacing
; LFirst specifies a style that starts a list
; LEnd specifies a non-list style that ends any prior lists
; LNest specifies a style that nests in an enclosing list
; LLevel specifies the nesting level to use, 1-30
; DListDD specifies use of dd instead of dt for items in dl lists
```

Note: **Mif2Go** overrides these settings with properties you specify in a format configuration file for the same formats; see §5.7.4 [Configuring list formats](#) on page 86.

For all List*n* styles:

- Assign LFirst to each paragraph format that starts a list, regardless of level or nesting, to restart numbering; see §21.12.2.2 [Converting lists with multiple paragraph formats](#) on page 676.
- Assign LEnd to each non-list paragraph format that immediately follows a list; see §21.12.2.2 [Converting lists with multiple paragraph formats](#) on page 676.

- Assign `LNest` and `LLeveln` to each paragraph format in a list that is nested inside another list; see §21.12.2.3 [Converting nested lists](#) on page 676.

You can assign more than one list keyword to a format. For example:

```
[HTMLParaStyles]
Numbered1 = List1 LFirst
Numbered = List1
Bulleted = List6
Body = LEnd
Heading* = LEnd
```

When you assign `Listn`, FrameMaker autonumbers are removed automatically; if the style you choose includes an HTML autonumber, that symbol is used instead, depending on browser interpretation. List styles 1 through 8 generally are reliable. List styles 9 through 12 are browser dependent; the same style in different browsers might show up with or without indents or bullets.

Note: Any format that can end a list must be assigned `LEnd`; otherwise the paragraphs that follow will be treated as part of the last list item, and will be indented.

If you are using CSS, **Mif2Go** applies the same class name used in the first `` item in a list to the `` or `` that precedes it. This permits convenient CSS adjustment of margins before and after lists, obviating the need to use distinct paragraph formats for the first and last list items.

You will need additional settings for the following:

- Lists that include more than one paragraph format; see §21.12.2.2 [Converting lists with multiple paragraph formats](#) on page 676.
- Nested lists; see §21.12.2.3 [Converting nested lists](#) on page 676.
- Dictionary-style lists; see §21.12.2.4 [Converting dictionary lists](#) on page 677.

21.12.2.2 Converting lists with multiple paragraph formats

If you use a different FrameMaker paragraph format for the first item in a list, and perhaps also for the last item, specify the appropriate `Listn` style for all of the paragraph formats; and also specify the following properties:

```
[HTMLParaStyles]
; LFirst specifies a style that starts a list
; LEnd specifies a non-list style that ends any prior lists
```

LFirst starts a list

Assign `LFirst` to the format that starts a list. If you do not assign `LFirst` to a format that can start a list, **Mif2Go** thinks an item in that format is being continued after a non-list item; if that is not the case, **Mif2Go** might be using a value that was never set.

*LEnd comes after
a list*

Do not assign `LEnd` to the format that ends a list; instead, assign `LEnd` to each paragraph format that can occur immediately *after* the end of a numbered list in your FrameMaker document. This means that you must assign `LEnd` to several non-list paragraph formats; this is the only way to get the indents right, without using CSS. The `LEnd` property can be annoying, because you must assign it to every format that could ever *end* a list, as opposed to being *included in* a list. To avoid unwanted left indents you must assign `LEnd` to *Body*, to all the headings, to figure titles and table anchors, and so forth.

21.12.2.3 Converting nested lists

If you used nested lists in your FrameMaker document, you must assign the following properties to inner list formats:

```
[HTMLParaStyles]
; LNest specifies a style that nests in an enclosing list
; LLevel specifies the nesting level to use, 1-30
```

Make each level a different list type

HTML does not let you nest a list inside another of the same type. If you have a bulleted list with a bulleted sublist, change the bulleted style for the sublist; for example, if the top-level list is `List6`, make the sublist style `List7`. Also make the top-level list `LLevel1`, and make the sublist both `LNest` and `LLevel2`. You can nest quite deeply and still retain the structure, if you apply these properties correctly.

Suppose your FrameMaker document has one numbered list nested inside another. You would assign `LLevel1` to the outer list format, and `LLevel2` to the inner (nested) list format. You would also assign the `LNest` property to the nested format, and, if you use a different format for the first item, assign `LFirst` to the first-item format in both lists. For example:

```
[HTMLParaStyles]
Numbered1 = List1 LLevel1 LFirst
Numbered = List1 LLevel1
AlphaSub1 = List4 LLevel2 LFirst LNest
AlphaSub = List4 LLevel2 LNest
```

If you are using CSS, you might want to add, in the CSS file:

```
ol ol {list-style-position: outside}
```

This is how to specify properties for nested lists in CSS.

If you are *not* using CSS, and your document has nested lists, you might need this setting:

```
[CSS]
; AlwaysNestLists = No (default, no nesting when CSS used) or Yes
AlwaysNestLists = Yes
```

However, if you use CSS at all for list items, you will get a mess if the lists really do nest. **Mif2Go** prevents that by default (with `AlwaysNestLists=No`), when you use class attributes. Setting `AlwaysNestLists=Yes` turns off this safety net, so you will have to adjust the CSS for the nested items to prevent overindenting. And also wave goodbye to cross-browser consistency.

21.12.2.4 Converting dictionary lists

For `List11` or `List12` (dictionary-style) lists, a `<dt>` tag is used for each term, normally flush left; and a `<dd>` tag for the definition, normally slightly indented. By default, **Mif2Go** puts out only `<dt>` item tags for `<dl>` lists. To specify `<dd>` tags also, assign property `DListDD` to the paragraph format you use for dictionary-style terms:

```
[HTMLParaStyles]
; DListDD specifies use of dd instead of dt for items in dl lists
GlosTerm = List12 DListDD
```

21.12.2.5 Including FrameMaker autonumbers in lists

When you use the HTML list styles, the browser itself supplies the bullet or autonumber; the one provided by FrameMaker is automatically dropped. To keep a FrameMaker bullet or autonumber, perhaps because you have chosen an unbulleted style (such as `List11` in some browsers):

```
[HTMLParaStyles]
; Anum includes Frame autonumber in list, default omits it
MyBulletStyle = Anum
```

21.12.2.6 Omitting CSS class attributes from list entries

To omit the CSS class attribute from list items:

```
[CSS]
; NoClassLists = Yes (default, no class in <li> tags), or No
; Default is reversed to No if UseCSS=Yes.
NoClassLists = Yes
```

If you use CSS, the default value of NoClassLists is reversed to No; see §22.5 [Understanding how CSS affects other options](#) on page 687.

21.12.2.7 Including or excluding the type list attribute

Three attributes apply to list wrappers (ol, ul) and list items (li): type, start, and value; the first two apply only to list wrappers:

- The type attribute specifies the kind of numbering, such as 1 or a.
- The start attribute specifies the starting value for the list; **Mif2Go** applies this attribute if lists are *not* nested, and a list is interrupted by another list. In that case, start tells the second (resumed) part of the first list where to restart numbering.
- The value attribute applies also to list items, and specifies the number for the current item.

Before CSS, this was how you controlled lists.

By default, **Mif2Go** includes the type attribute in list wrappers ol and ul. To omit the type attribute from list wrappers:

```
[CSS]
; NoAttribLists = No (default, use type, start, and value attributes
; in list tags), or Yes (omit type attribute from list tags)
NoAttribLists = Yes
; UseListTypeAttribute = Yes (default for JavaHelp, to fix CSS bug)
; or No (default for other formats, go by NoAttribLists value)
UseListTypeAttribute = Yes
```

Note: If you use both **Mif2Go** and **DITA2Go**, be aware that the default for NoAttribLists is Yes for **DITA2Go**.

When NoAttribLists=No, all three attributes are allowed on list tags.

When NoAttribLists=Yes, the value and start attributes are allowed, and use of the type attribute is left up to the value of UseListTypeAttribute, which defaults to No (meaning leave it up to NoAttribLists) except for JavaHelp and Oracle help, both of which need the type attribute.

21.12.3 Indenting list items

To consistently indent the second and subsequent lines of a bulleted or numbered item so the text more or less lines up with the start of the first-line text, you have two choices.

Neither method is precise:

§21.12.3.1 [Adjusting the second-line list indent in CSS](#) on page 678

§21.12.3.2 [Inserting spaces between first-line list autonumber and text](#) on page 679

21.12.3.1 Adjusting the second-line list indent in CSS

You can use CSS to adjust the second-line indent of a list format to match the first line. Using CSS is tricky, because CSS1 provides no equivalent of the FrameMaker autonumber-tab-hang construct; the tab concept is missing from CSS. Therefore, spacing between the bullet or autonumber and the text is browser dependent.

To use CSS with `` or `` tags, you must deduct from your CSS indent the amount of indent applied automatically by the browser, or you will see your `` items sliding away to the right like elections in Florida. How much you deduct is browser dependent, so you cannot have one CSS for all browsers. Instead you must play the JavaScript detection game to select among multiple CSS files at run time. Although **Mif2Go** supports this method and provides rudimentary JavaScript (see §22.6.1 [Selecting a CSS file at run time](#) on page 688), this is not a standards-friendly approach.

21.12.3.2 Inserting spaces between first-line list autonumber and text

You can use macros to insert fixed spaces after the autonumber or bullet to get the first text line to align with subsequent lines. This is not a precise method, because you can adjust space only in nbsp-width increments.

To add fixed spaces after an autonumber or bullet:

```
[HTMLParaStyles]
; Paragraph format = keywords for functions and properties
ListFormat = CodeAfterAnum

[AnumCodeAfter]
; doc style = HTML code to use after end of autonumber sequence
ListFormat = &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
```

You have to determine the proper number of spaces by trial and error. And sadly, it will be different when you get to double digits in numbered lists: then you must reduce the number of `nbsps` by one. You could maintain a macro-variable counter (see §28.3 [Using macro variables](#) on page 795), but synchronizing the count with the FrameMaker numbering would be challenging.

21.12.4 Converting list formats to HTML/XML paragraphs

Mif2Go can map list formats to <p> items. CSS preserves the numbering used in your FrameMaker document, no matter how complex. If you allow **Mif2Go** to create a CSS file for your document (see §22 [Setting up CSS for HTML](#) on page 681), **Mif2Go** uses CSS to reproduce the original indents.

Some things to consider about converting lists:

- You *must* use this method for list formats with complex autonumbers that have no counterpart in HTML; for example, the multi-level section numbers in the **Mif2Go User's Guide** (this document).
- Because autonumbers are converted to text with this method, you cannot insert new items in a numbered list in the HTML output (something you should not be doing anyway) without manually renumbering the rest of the list.
- When you convert lists from FrameMaker to regular paragraphs, rather than use HTML list styles, by default **Mif2Go** retains the FrameMaker autonumbers (including bullets). To remove them, see §21.3.3 [Converting paragraph formats with autonumbers](#) on page 648.

22 Setting up CSS for HTML

Much of what used to be set by attributes in HTML is now better handled in CSS (Cascading Style Sheets). This section shows how to use and customize CSS for HTML output. Topics include:

- §22.1 [Deciding whether to use CSS](#) on page 681
- §22.2 [Understanding how to use CSS](#) on page 681
- §22.3 [Understanding how Mif2Go generates CSS](#) on page 682
- §22.4 [Specifying CSS file and link options](#) on page 683
- §22.5 [Understanding how CSS affects other options](#) on page 687
- §22.6 [Linking to alternate CSS files](#) on page 688
- §22.7 [Assigning CSS classes](#) on page 691
- §22.8 [Customizing CSS properties](#) on page 698

22.1 Deciding whether to use CSS

With respect to CSS style sheets for your project, you can do any of the following:

- Specify an existing style sheet for **Mif2Go** to use.
- Have **Mif2Go** create a new style sheet based on formats and configuration settings.
- Select a style sheet at run time, according to the browser in use.
- Choose not to use CSS at all.

If the HTML output you produce will be viewed with a browser that offers CSS support (which most Web browsers do these days), CSS is the better way to manage presentation, compared to `` tags and such. On the other hand, CSS is implemented somewhat inconsistently among different browsers. You might have to provide several cascading style sheets, to be automatically selected from at run time; and you can spend an amazing amount of time tuning style sheets and adding JavaScript macros.

Note: Formatting that is directly created by an HTML tag overrides CSS. Using HTML presentational tags and attributes cripples your ability to use CSS, and therefore to adjust formatting easily without having to alter content.

If you are creating in-house HTML documents, use whatever works with local browsers.

Note: If you are creating XHTML output for the Web, Netscape Navigator 7 and Mozilla ignore your CSS files.

The default is for **Mif2Go** to use CSS for standard HTML and for HTML-based Help, and to create a style sheet for you, based on the formats in your FrameMaker document; see §22.4 [Specifying CSS file and link options](#) on page 683.

To look at your page in different browsers, using a Web-based method:

<http://www.anybrowser.com/>

See also:

- §22.5 [Understanding how CSS affects other options](#) on page 687

22.2 Understanding how to use CSS

If you are not familiar with CSS, here are some good starting points, tutorials, and reference sites:

<http://websites tips.com/css/index.shtml>
<http://www.mako4css.com/Tutorial.htm>
<http://www.w3schools.com/css/>
<http://www.thenoodleincident.com/tutorials/index.html>
<http://www.alistapart.com/>

Also the spectacular and inspiring:

<http://www.csszengarden.com/>

And a few books:

Cascading Style Sheets: The Definitive Guide, by Eric A. Meyer

The basics of creating, saving, formatting, and linking to CSS: HTML for the World Wide Web with XHTML and CSS, by Elizabeth Castro

Cascading Style Sheets: Designing for the Web, by Hakon Lie and Bert Bos

(**Note:** Hakon Lie is the W3C designer of CSS)

22.3 Understanding how Mif2Go generates CSS

Mif2Go generates CSS based on the formats in your FrameMaker document. To get the precise display you want, you might find that you need to edit the resulting style sheet in a text editor or a CSS editor. **Mif2Go** excludes most font tags, and optionally excludes typographic tags, because those can override CSS. What you get is very clean HTML, with @class attributes.

Although the default is to create a new CSS each time you run a conversion, there is a setting you can specify to retain the CSS *as is*; see §22.4 [Specifying CSS file and link options](#) on page 683. That is what we usually advise people to do if they customize the CSS. The downside is that if you define new formats (classes, in HTML) for your document, **Mif2Go** cannot add them to the CSS; you have to do that yourself.

By default, the first time you convert a document to HTML or XML, or generate HTML-based Help, **Mif2Go** creates a style sheet for the output. After importing any conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67), **Mif2Go** creates a new CSS file that contains all the catalogued paragraph and character format names from your FrameMaker document, based on the following:

- whatever CSS classes you assign to those formats in [ParaClasses] or [CharClasses]; see:
 - §22.7.2 [Mapping paragraph formats to CSS classes](#) on page 692
 - §22.7.3 [Mapping character formats to tags or span classes](#) on page 693
- whatever tags you set for those formats in [ParaTags] and [CharTags]; see:
 - §21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646
 - §21.4 [Mapping character formats](#) on page 653.

Absent explicit CSS settings in [ParaClasses], [CharClasses], [ParaTags], or [CharTags], **Mif2Go** bases CSS class names on your FrameMaker format names, possibly reduced to fit CSS naming rules for class names; see §22.7.1 [Understanding CSS class name restrictions](#) on page 691.

CSS rendition is affected by mappings in the following sections:

[Fonts] (see §21.7.2 [Remapping fonts](#) on page 664)

[Colors] (see §21.9 [Specifying text colors for HTML](#) on page 669)

However, all other CSS properties come directly from FrameMaker paragraph and character catalogs. FrameMaker markers do not affect CSS entries.

See §22.7 [Assigning CSS classes](#) on page 691.

22.4 Specifying CSS file and link options

In this section:

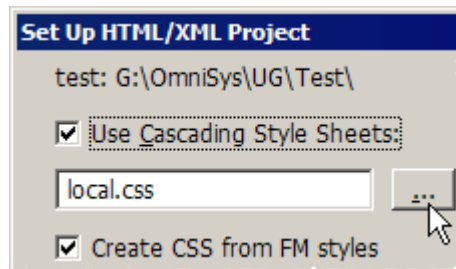
- §22.4.1 [Specifying CSS options at project set-up time](#) on page 683
- §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684
- §22.4.3 [Designating and locating a CSS file](#) on page 686
- §22.4.4 [Directing Mif2Go to generate a CSS file](#) on page 686
- §22.4.5 [Understanding effects of the older Stylesheet setting](#) on page 687

22.4.1 Specifying CSS options at project set-up time

When you start a *new* HTML or XML project, you can specify some CSS file options at set-up time; see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425. (For HTML-based Help, you must specify these options in a configuration file.)

[Figure 22-1](#) shows the CSS portion of the *Set Up HTML/XML Project* dialog. However, once you have set up your project, to change any of these CSS options you must edit the [CSS] section of the configuration file; see §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684.

Figure 22-1 CSS set-up options



- | | |
|-------------------------------|--|
| <i>Use CSS</i> | To use CSS for your output, check Use Cascading Style Sheets . By default, Mif2Go looks in the project directory for a CSS file named <code>local.css</code> . You can specify a different file name, or browse for a different file and location. To change this option in the configuration file, see §22.4.3 Designating and locating a CSS file on page 686. |
| <i>Create CSS</i> | To have Mif2Go generate a CSS file for you based on the formats in your document, check both Use Cascading Style Sheets and Create CSS from FM styles . By default, Mif2Go names the new CSS file <code>local.css</code> and places it in the project directory. You can specify a different file name and location. To change this option in the configuration file, see §22.4.4 Directing Mif2Go to generate a CSS file on page 686. |
| <i>Select CSS at run time</i> | To defer CSS file selection until run time, uncheck Create CSS from FM styles , and check only Use Cascading Style Sheets . However, <i>you must also edit settings in the configuration file</i> ; see §22.4.2 Specifying CSS options in a Mif2Go configuration file on page 684. You cannot fully specify this option in the <i>Set Up</i> dialog. When you defer CSS file selection, Mif2Go ignores whatever was specified in the <i>Set Up</i> dialog for CSS file name and location. See §22.6.1 Selecting a CSS file at run time on page 688. |
| <i>Skip CSS</i> | To avoid using CSS at all, uncheck both checkboxes. |
| <i>Accept defaults</i> | By default, when you set up a <i>new</i> HTML or XML project, Mif2Go does the following: <ul style="list-style-type: none"> • Specifies that CSS will be used for the output. • Names the CSS file <code>local.css</code>. |

- Creates `local.css` in the project directory, if a file of that name is not already present.
- Bases the content of `local.css` on the formats in your FrameMaker document.

If you make *no changes* in the *Set Up HTML/XML Project* dialog to the options shown in [Figure 22-1](#), your configuration file will contain the following values:

```
[CSS]
UseCSS=Yes
WriteCssStylesheet=Once
CssFileName=local.css
```

22.4.2 Specifying CSS options in a Mif2Go configuration file

To specify CSS options in a **Mif2Go** configuration file:

```
[CSS]
; UseCSS = Yes (default) or No
UseCSS=Yes
; WriteClassAttributes = Yes (default)
; or No (when ClassIsTag=Yes or when not using CSS)
WriteClassAttributes=Yes
; WriteCssStylesheet = Once (default), Always, or Never
WriteCssStylesheet=Once
; WriteCssLink = Yes (default) or No
WriteCssLink=Yes
; CssBrowserDetect= Macro reference to JavaScript code that determines
; browser type and writes link from HTML to appropriate CSS file
;CssBrowserDetect=<$BrowserCSS>
; CssFileName = name of style sheet to reference (file name, no path)
CssFileName=local.css
```

Use these options to do the following:

[Direct Mif2Go to use CSS](#)

[Include class attributes](#)

[Designate a CSS file](#)

[Create a CSS file](#)

[Link to a CSS file](#)

[Select a CSS file at run time](#)

See also §Table 22-2 [CSS-dependent default values of options](#) on page 688.

Note: If you have been using `[HtmlOptions]Stylesheet` to specify CSS file options, see §22.4.5 [Understanding effects of the older Stylesheet setting](#) on page 687. The `Stylesheet` setting is deprecated in favor of the `[CSS]` settings listed in this section.

*Direct **Mif2Go** to
use CSS*

To direct **Mif2Go** to use CSS for your output:

```
[CSS]
; UseCSS = Yes (default) or No
UseCSS=Yes
```

When `UseCSS=Yes`, by default **Mif2Go** does the following:

- includes class attributes in paragraph tags
- creates the CSS file designated by `CssFileName`, if this file is not already present
- includes a link from the `<head>` element of each output file to the CSS file designated by `CssFileName`.

When `UseCSS=No`, paragraph tags do not include class attributes, no CSS file is referenced in the output, and the remaining `[CSS]` options are ignored.

See also §22.5 [Understanding how CSS affects other options](#) on page 687.

<i>Include class attributes</i>	<p>WriteClassAttributes values have the following effects:</p> <p>Yes Mif2Go includes CSS class attributes in the paragraph tags in your output; see §22.3 Understanding how Mif2Go generates CSS on page 682.</p> <p>No Class attributes are not included in paragraph tags. Use this setting for XML output when [CSS]ClassIsTag=Yes, the default for XML; see §14.4.2 Deriving XML tags from format and class names on page 462.</p>
<i>Designate a CSS file</i>	<p>CssFileName designates the CSS file Mif2Go optionally creates and references. The default is local.css, located in the project directory. You can specify a different name and location for this file; see §22.4.3 Designating and locating a CSS file on page 686.</p>
<i>Create a CSS file</i>	<p>WriteCssStylesheet values have the following effects:</p> <p>Once Mif2Go creates a new CSS file based on your FrameMaker formats, but only if no CSS file of the name designated by CssFileName is already present in the project directory. This is the default Mif2Go puts in place at set-up. Specify Once to get a starting CSS file that you can tweak manually. See §22.4.4 Directing Mif2Go to generate a CSS file on page 686.</p> <p>Always Mif2Go creates a new CSS file based on your FrameMaker formats, overwriting in the project directory any existing CSS file of the name designated by CssFileName. Specify Always if you do not need to tweak the CSS file, or if you can make any needed changes in macros, either in the configuration file or in a macro library. See §22.4.4 Directing Mif2Go to generate a CSS file on page 686. You can specify additional settings to govern what Mif2Go includes in a CSS file; see §22.8.4 Overriding styles in Mif2Go-generated CSS files on page 700.</p> <p>Never Mif2Go does not create a new CSS file, nor overwrite an existing file. When UseCSS=Yes, Mif2Go assumes you wish to use an existing CSS file: either the file designated by CssFileName, or a file to be selected at run time, depending on the values of WriteCssLink and CssBrowserDetect. Specify Never if you want to use an existing CSS file. See §22.4.3 Designating and locating a CSS file on page 686.</p>
<i>Link to a CSS file</i>	<p>WriteCssLink values have the following effects:</p> <p>Yes If CssBrowserDetect is not present, Mif2Go includes in the <head> element a simple link to the CSS file designated by CssFileName, in the relative directory designated by CssPath. The link is one of the following types:</p> <p>For HTML:</p> <pre><link rel="stylesheet" href="local.css" type="text/css"></pre> <p>For XML:</p> <pre><?xml:stylesheet href="local.css" type="text/css" charset="UTF-8"?></pre> <p>If CssBrowserDetect is present, instead of the simple link Mif2Go includes the macro assigned to CssBrowserDetect in the <head> element. See §22.6.1 Selecting a CSS file at run time on page 688.</p> <p>No Mif2Go does not create a link to a CSS file. Use this setting when you are not using CSS, or when you provide your own macro in [Inserts]Head to select a CSS file dynamically, independently of CssBrowserDetect. See §22.6.1 Selecting a CSS file at run time on page 688.</p>

Select a CSS file at run time When a macro is assigned to `CssBrowserDetect`, if `WriteCssLink=Yes`, the macro is included in the `<head>` element. If `WriteCssLink=No`, the macro is ignored. See §22.6.1 [Selecting a CSS file at run time](#) on page 688.

22.4.3 Designating and locating a CSS file

To specify CSS file name and location in the configuration file:

```
[CSS]
UseCSS=Yes
; CssFileName = name of style sheet to reference in link when
; WriteCssLink=Yes and CssBrowserDetect is absent.
CssFileName=MyStyles.css
; CssPath = directory in which .css (or .xsl) files are to be placed
CssPath=./css
```

`CssFileName` designates the CSS file to be referenced when `WriteCssLink=Yes`. Do not include a path; the value of `CssFileName` should be just a file name with extension. The default value is `local.css`, and the default location is the directory designated by `CssPath`.

`CssPath` designates the directory to be referenced in CSS links when `WriteCssLink=Yes`. If you use backslashes in the path name, **Mif2Go** changes them to forward slashes before writing the path to your HTML output files. The default value of `CssPath` is the directory designated by `WrapPath`; see §35.8 [Placing CSS or XSL files for assembly](#) on page 969. You can have **Mif2Go** copy CSS files to the `CssPath` directory from another location at run time.

Path to CSS file should be relative If you specify a value for `CssPath`, the path should be relative to the directory containing your HTML files. *If you specify an absolute path, the CSS file is likely to be accessible only on your own machine.*

Default CSS file name and location If all of the following are true, **Mif2Go** creates a CSS file named `local.css` and places it in the project directory:

- You have indicated that you want **Mif2Go** to use CSS (that is, `UseCSS=Yes`).
- The value of `WriteCssStylesheet` is either `Once` or `Always`.
- The configuration file has *no entries at all* for either `CssFileName` or `CssPath`.
- The project directory does not already contain a file named `local.css`.

22.4.4 Directing Mif2Go to generate a CSS file

The first time you convert files for a project, if you intend to use CSS, probably you will want **Mif2Go** to generate a new CSS file, so you can use a style sheet that contains the equivalents of your FrameMaker format settings. You should also specify a name for the CSS file (see §22.4.3 [Designating and locating a CSS file](#) on page 686); for example:

```
[CSS]
UseCSS=Yes
WriteCssStylesheet=Once
CssFileName=MyStyles.css
```

When `WriteCssStylesheet=Once`, **Mif2Go** generates a new CSS file, but only if no CSS file of the same name (in this example, `MyStyles.css`) already exists in the project directory. This is probably the best setting to use in most circumstances; you can leave this setting in place, and any changes you make directly to the CSS file will be preserved the next time you run the conversion. On the other hand, changes you make to FrameMaker formats will *not* be reflected in the CSS file.

Force a new CSS, update CSS from formats

To force **Mif2Go** to generate a new CSS file, overwriting any existing CSS file of the same name in the project directory:

```
[CSS]
WriteCssStylesheet=Always
```

If you never make changes directly to the CSS file, you can let **Mif2Go** generate a CSS file each time; then any changes you make to your FrameMaker formats are updated automatically in the CSS file.

Update CSS directly

If you make changes directly to the CSS file, to prevent your changes from being overwritten, for subsequent conversion runs *you must change this setting* to Once or to Never:

```
[CSS]
WriteCssStylesheet=Never
```

Styles based on configuration settings

You can have it both ways, by specifying CSS settings in your configuration file for particular formats; see §22.8.4 [Overriding styles in Mif2Go-generated CSS files](#) on page 700

Styles based on import template

If you direct **Mif2Go** to import a conversion template (see §30.7 [Applying FrameMaker conversion templates](#) on page 863), the properties in the generated CSS file will be those in effect after the import.

22.4.5 Understanding effects of the older Stylesheet setting

Prior versions of **Mif2Go** used a single setting to manage CSS file options:

```
[HtmlOptions]
; Stylesheet = None (default if no setting),
; Init (default set by Setup, write .css if not existing),
; Generate (overwrite .css),
; Class (no link, no write), or
; Use (link, no write).
; Stylesheet=None
```

Stylesheet is deprecated

The `Stylesheet` setting is deprecated, and is replaced by the `[CSS]` options described in §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684. However, `Stylesheet` is still supported for backward compatibility.

If a `Stylesheet` setting is present in your configuration file and the newer `[CSS]` file options are not present, defaults for the newer options are set according to the value of `Stylesheet`, as shown in [Table 22-1](#). If both are present, the `[CSS]` options prevail.

Table 22-1 Default CSS file options when `[HtmlOptions]Stylesheet` is used

[CSS] option	[HtmlOptions] Stylesheet setting				
	None	Init	Generate	Class	Use
UseCSS	No	Yes	Yes	Yes	Yes
WriteClassAttributes	No	Yes	Yes	Yes	Yes
WriteCssStylesheet	Never	Once	Always	Never	Never
WriteCssLink	No	Yes	Yes	No	Yes

22.5 Understanding how CSS affects other options

The choice to use CSS changes the behavior of certain other options. Be aware of the following:

[Using CSS turns off tags by default](#)

Using CSS changes some default values

Not using CSS changes other default values.

Using CSS turns
off tags by
default

When UseCSS=Yes, the default value for [HTMLOptions]NoFonts results in removal of tags; these tags are deprecated, and are not needed with CSS.

However, if you specify UseCSS=Yes and then do *not* supply or create a CSS file, text appearance in your HTML output still depends on tags, which are no longer present unless you explicitly set [HTMLOptions]NoFonts=No in the configuration file. Therefore, the text in your document might not come out the way you expect.

See §21.7.4 [Including or excluding font tags](#) on page 665.

Using CSS
changes some
default values

When UseCSS=Yes, default values are reversed for the [Graphics] and [HtmlOptions] settings listed in [Table 22-2](#). This removes most HTML that can interfere with CSS settings. (One exception is [HTMLOptions]AllowOverrides, which defaults to Yes for HTML (but not for XML or DITA) to retain any incidental use of bold and italic in text.)

Not using CSS
changes other
default values

When UseCSS=No, default values are reversed for the [CSS] settings listed in [Table 22-2](#).

Table 22-2 CSS-dependent default values of options

Section	Option	Default value of option when:		
		UseCSS=Yes	UseCSS=No	Ref.
[CSS]	LinkClassIsParaClass	Yes	No	19.2.2.2
	NoClassLists	No	Yes	21.12.2.6
	WriteClassAttributes	Yes	No	22.4.2
	WriteCssStylesheet	Once	Never	22.4.2
	WriteCssLink	Yes	No	22.4.2
	XrefFormatIsXrefClass	Yes	No	22.7.8
[Graphics]	GraphAlignAttributes	No	Yes	23.6.2
[HtmlOptions]	AlignAttributes	No	Yes	21.5
	Basefont	No	Yes	21.7.1
	NoFonts	Yes	No	21.7.4

22.6 Linking to alternate CSS files

In this section:

§22.6.1 [Selecting a CSS file at run time](#) on page 688

§22.6.2 [Changing CSS files in the middle of a document](#) on page 689

§22.6.3 [Customizing the CSS link tag](#) on page 690

§22.6.4 [Using an alternate CSS link tag for Netscape 4](#) on page 690

22.6.1 Selecting a CSS file at run time

CSS support is a mixed bag; a lot depends on exactly which browsers, and which versions of them, you need to support. You might need to autodetect the browser and choose from different CSS files at run time, using a macro instead of a fixed link, to reference JavaScript code that detects the type of browser in use and selects an appropriate CSS file. For example:

```
[CSS]
WriteCssLink=Yes
CssBrowserDetect=<${SelectCSS1}>
```

As an alternative:

```
[CSS]
WriteCssLink=No

[Inserts]
Head=<${SelectCSS1}>
```

Provide the referenced macro:

```
[SelectCSS1]
; Include here the JavaScript from m2hmacro.ini
```

Sample macro [\${SelectCSS1}] contains JavaScript to detect several popular browsers. This macro, and an alternate, [\${SelectCSS2}], are included in file `m2hmacro.ini`, in your **Mif2Go** distribution directory. You can copy `m2hmacro.ini` file to the project directory, or just copy the macro definition into the configuration file for your project; see §28.1.1.2 [Understanding where you can define named macros](#) on page 788. You can modify the macro definition as needed; consult a JavaScript reference for syntax.

22.6.2 Changing CSS files in the middle of a document

To have **Mif2Go** reference different style sheets for output from different parts of your FrameMaker document, you can use a macro to provide the CSS link, then insert markers in your document to signal a change of CSS file. To prevent **Mif2Go** from automatically generating a CSS file reference, you must also specify:

```
[CSS]
WriteCssLink=No
```

To generate a CSS file reference from your document, assign a macro to be placed in the `<head>` element of each HTML output file; for example:

```
[Inserts]
Head=<${CSSmacro}>
```

Include in the macro definition a macro variable (for example, `$$myAltCSS`) in place of the base name of the CSS file:

```
[CSSmacro]
; You must type the following all on one line:
<link rel='stylesheet' href='<$$myAltCSS>.css' charset=ISO-8859-1
type='text/css' />
```

Give the macro variable an initial value: the base name of the first CSS file you want referenced:

```
[MacroVariables]
myAltCSS=UsualCSS
```

Mif2Go uses the value of macro variable `$$myAltCSS` to select a CSS file at the start of each file split.

To change `$$myAltCSS` to a different value for a subsequent split, you must place a marker in a paragraph *before* the split. You can use a FrameMaker **HTML Macro** marker, with content as follows:

```
<$$myAltCSS=OtherCSS>
```

Or, you could create a new marker type (for example, **CSSname**; see §29.2 [Adding custom marker types](#) on page 832), and provide as content only the base name of the CSS file:

```
OtherCSS
```

To assemble the macro around the CSS file value, also specify the following:

```
[MarkerTypes]
CSSname=Code

[MarkerTypeCodeBefore]
CSSname=<$$myAltCSS=

[MarkerTypeCodeAfter]
CSSname= >
```

To change the value of `$$myAltCSS` for a particular FrameMaker file in your document, place in the project directory a file-specific configuration file that contains (only) the following setting:

```
[MacroVariables]
myAltCSS=SpecialCSS
```

See §33.1 [Using a different configuration for selected files](#) on page 919.

22.6.3 Customizing the CSS link tag

The generic CSS link tag **Mif2Go** inserts in your HTML output looks like this:

```
<link rel="stylesheet" href="MyStyles.css" type="text/css">
```

Suppose you want to specify additional properties for the CSS file, such as media type. First, you must prevent **Mif2Go** from writing the generic link tag:

```
[CSS]
WriteCssLink=No
```

To specify the link yourself, assign it to the `<head>` element:

```
[Inserts]
; You must type the following all on one line:
Head=<link rel="stylesheet" href="MyStyles.css" type="text/css"
media="screen">
```

As an alternative, you could reference the link as a macro (see §28.1 [Defining and invoking macros](#) on page 787):

```
[Inserts]
Head=<MyCSSLink>

[MyCSSLink]
<link rel="stylesheet" href="MyStyles.css" type="text/css"
media="screen">
```

You could even go a step further, and provide a macro variable (see §28.3 [Using macro variables](#) on page 795) for the value of the attribute, so you can change the value in just one place:

```
[MyCSSLink]
<link rel="stylesheet" href="MyStyles.css" type="text/css"
media="<$$MediaType>">

[MacroVariables]
MediaType=screen
```

22.6.4 Using an alternate CSS link tag for Netscape 4

Netscape Navigator 4.x plays better with CSS if the link to the style sheet specifies `type="text/css1"` instead of `type="text/css"`:

```
<link rel="stylesheet" href="MyStyles.css" type="text/css1">
```

For example, tags ` . . . ` are ignored if the CSS entry for the class does not specify `bold`. If your HTML output will be viewed with Netscape Navigator 4.x, you can include the alternate link with this option:

```
[CSS]
; CSSLinkNS4 = No (default, required for CSS validation)
; or Yes (NS 4.x)
CSSLinkNS4=Yes
```

Unfortunately, this breaks the W3C CSS Validator, which claims there is no style sheet.

22.7 Assigning CSS classes

In this section:

- §22.7.1 [Understanding CSS class name restrictions](#) on page 691
- §22.7.2 [Mapping paragraph formats to CSS classes](#) on page 692
- §22.7.3 [Mapping character formats to tags or span classes](#) on page 693
- §22.7.4 [Assigning CSS classes to table formats](#) on page 694
- §22.7.5 [Assigning CSS classes to text and table footnotes](#) on page 694
- §22.7.6 [Assigning CSS classes based on Unicode character ranges](#) on page 694
- §22.7.7 [Assigning CSS classes to FrameMaker conditions](#) on page 695
- §22.7.8 [Using link format names as CSS class names](#) on page 696
- §22.7.9 [Using CSS class names as tags for XML](#) on page 696
- §22.7.10 [Omitting tags from CSS selectors](#) on page 696
- §22.7.11 [Overriding CSS class for selected paragraphs](#) on page 697

See also:

- §21.12.2.7 [Including or excluding the type list attribute](#) on page 678

22.7.1 Understanding CSS class name restrictions

*Use only letters
and numbers in
class names*

Class names used with CSS may contain alphanumeric characters only. You cannot use spaces or symbols; not even underscores. Class names in HTML output must match in case the same names in the CSS file. **Mif2Go** imposes an internal limit of 128 characters on CSS class names.

To create class names from format names, **Mif2Go** does the following:

- removes or replaces spaces
- removes all non-alphanumeric characters
- replaces accented characters with their non-accented equivalents; see §21.6.4 [Converting Western European accented characters](#) on page 660
- for output types that require lowercase CSS, changes all characters to lowercase, regardless of whether format names are uppercase, lowercase, or mixed case.

These transformations might lead to conflicts if your format names differ only in spacing, in case, or by any removed characters. See §2.2 [Naming FrameMaker formats](#) on page 66.

*Replace spaces
with a character*

You can specify a letter, a number, an underscore, or a hyphen to substitute for spaces in class names. For example:

```
[HtmlOptions]
; These alphanumeric chars are used as space replacements in IDs;
; if non-alphanumeric (other than hyphen or underscore), spaces are
; stripped instead (default)
; ClassSpaceChar = char to use as space replacement
ClassSpaceChar = _
```

- Remove spaces* By default, **Mif2Go** removes spaces without replacing them. The same thing happens if you set `ClassSpaceChar` to any non-alphanumeric character other than a hyphen or an underscore: **Mif2Go** removes all spaces without replacing them.
- Case of class names* CSS does not distinguish between names that differ only in case; if you use both *heading1* and *Heading1*, and they are defined differently, you are sure to see some unexpected results. Class names in HTML files must match in case the corresponding names in the CSS file. Class names can be mixed case for some output types, but must be lowercase for other output types:
- For XML, XHTML, JavaHelp, and Oracle Help, **Mif2Go** changes all generated class names in output in lowercase.
 - For standard HTML, HTML Help, and OmniHelp, class names generated from FrameMaker formats retain their original case.

You can force lowercase class names for any HTML output type. To make generated class names all lowercase:

```
[CSS]
; LowerCaseCSS = No (default mixed case)
; or Yes (lower case only, JH, OHJ, XML, and XHTML)
LowerCaseCSS = Yes
```

22.7.2 Mapping paragraph formats to CSS classes

When you use CSS, by default **Mif2Go** maps each FrameMaker paragraph format name to a CSS class of the same name, applying to the name any needed transformations (see §22.7.1 [Understanding CSS class name restrictions](#) on page 691).

For a paragraph format, by default the class name in the **Mif2Go**-generated CSS file is preceded by the tag name and a dot:

tagname.classname

The tag name comes from whatever is specified for that format in `[ParaTags]` (see §21.3 [Mapping paragraph formats](#) on page 646), or else `<p>`. Unless you assign classes explicitly, the class name is based on the FrameMaker paragraph format name.

For example, suppose your FrameMaker document includes catalogued paragraph formats *Chap_Title*, *Heading*, and *Body*, with the first two assigned HTML tags in `[ParaTags]`. **Mif2Go** would treat these formats as follows, provided `ClassIsTag=No` (see §22.7.9 [Using CSS class names as tags for XML](#) on page 696):

<u>FM format name</u>	<u>[ParaTags]</u>	<u>Mif2Go HTML output</u>	<u>Mif2Go CSS entry</u>
<i>Chap_Title</i>	Chap_Title=H1	<code><h1 class="chaptitle"></code>	<code>h1.chaptitle {...}</code>
<i>SubHead</i>	SubHead=H2	<code><h2 class="subhead"></code>	<code>h2.subhead {...}</code>
<i>Body</i>	(no setting)	<code><p class="body"></code>	<code>p.body {...}</code>

Mif2Go includes as many of the following properties as apply, based on the format properties in your document (as modified by any imported conversion template), for each paragraph format (class) in the CSS file:

```
font: [ italic | small-caps | bold ]
margin: top right bottom left
text-align: [ center | right ]
text-indent: [ for first line, negative for hang ]
text-decoration: [ underline | line-through ]
text-transform: [ uppercase | lowercase | capitalize ]
color: #RRGGBB
```

To explicitly map individual FrameMaker paragraph format names to CSS class names:

```
[ParaClasses]
; Document style name = class to use (default is based on name)
; For XML, the class is used as the tag name by default.
FormatName=classname
```

Or:

```
[ParaTags]
FormatName= class="classname"
```

If you assign class names to the same format in both [ParaClasses] and [ParaTags], and the class names are different, **Mif2Go** uses the [ParaTags] setting for backward compatibility. See §21.3.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 646.

*Anchor paragraph
class*

If your document uses a special paragraph format to anchor graphics, you can specify a class name for the anchor format:

```
[Graphics]
; GraphClass = class name to use for paras created to hold <img> tags
GraphClass=graphic
```

XML For XML output, see §14.4.2 [Deriving XML tags from format and class names](#) on page 462.

22.7.3 Mapping character formats to tags or span classes

When you use CSS, **Mif2Go** generates any tags assigned to a character format in [CharTags]; see §21.4 [Mapping character formats](#) on page 653. By default, **Mif2Go** maps each FrameMaker character format that is *not* assigned a tag in [CharTags] to a CSS span class of the same name as the format, applying to the name any needed transformations (see §22.7.1 [Understanding CSS class name restrictions](#) on page 691).

For example, suppose your FrameMaker document uses catalogued character format names *Emphasis*, *Prog Term*, and *Link*, with the first two assigned HTML tags in [CharTags]. **Mif2Go** would treat these formats as follows, provided ClassIsTag=No (see §22.7.9 [Using CSS class names as tags for XML](#) on page 696):

<u>FM format name</u>	<u>[CharTags]</u>	<u>Mif2Go HTML output</u>	<u>Mif2Go CSS entry</u>
<i>Emphasis</i>	Emphasis=em		em.emphasis {...}
<i>Prog Term</i>	Prog Term=code	<code>	code.progterm {...}
<i>Link</i>	(no setting)		span.link {...}

If no tags are specified in [CharTags] for a particular character format, by default that format gets a span class.

To avoid creating CSS span classes for any character formats that are neither explicitly assigned an HTML tag nor explicitly assigned to a span class:

```
[CSS]
; UseSpanAsDefault = Yes (default, use span as element name
; for all char formats that do not specify one in [CharTags]
; or No
UseSpanAsDefault=No
```

When UseSpanAsDefault=Yes, any catalogued character format name not listed in [CharTags] is assigned to a span class of the same name as the format.

When UseSpanAsDefault=No, any catalogued character format name not listed in [CharTags] is skipped, and becomes just an override in HTML output.

Untagged *Bold* and *Italic* applied with FrameMaker toolbar buttons get mapped to `` and `<i>` respectively.

To explicitly map an individual character format to a CSS span class:

```
[CharTags]
CharFormat=span

[CharClasses]
CharFormat=classname
```

Or:

```
[CharTags]
CharFormat=span class="classname"
```

You can use either method to assign `` tags, to define character formats globally in CSS. For example, if you map character format *CodeBold* to ``, **Mif2Go** inserts corresponding generic selector `.codebold` in the CSS file.

If you assign a class name to the same format in both `[CharClasses]` and `[CharTags]`, and the class names are different, **Mif2Go** uses the `[CharTags]` setting for backward compatibility. See §21.4 [Mapping character formats](#) on page 653.

Generic XML For generic XML output, see §14.4.2 [Deriving XML tags from format and class names](#) on page 462.

22.7.4 Assigning CSS classes to table formats

To explicitly map individual FrameMaker table format names to CSS class names:

```
[TableClasses]
; Table format name = class to use (default is based on name)
; For XML, the class is used as the tag name by default.
TableFormatName = classname
```

See also:

§24.4.3 [Assigning a CSS class to a table](#) on page 737

22.7.5 Assigning CSS classes to text and table footnotes

To assign CSS classes to text footnotes and to table footnotes:

```
[CSS]
; FootClass = name for CSS class for footnotes, default "footnote"
FootClass = footnote
; TbFootClass = name to use for CSS class for table footnotes
TbFootClass = tablefootnote
```

See also:

§21.11 [Converting footnotes to HTML or XML](#) on page 671

§24.5.2 [Configuring and positioning table titles](#) on page 747

22.7.6 Assigning CSS classes based on Unicode character ranges

Suppose your document is translated to a non-Western language: Japanese, for example. After translation, a certain number of words might remain in Latin characters: product names, feature names, and acronyms, for example. The glyphs for Latin characters in common Unicode fonts (such as Mincho) that include Japanese characters might be unacceptably ugly. What you need is an automatic way to specify a different font to use for those glyphs.

Mif2Go provides settings that allow you to assign a CSS class to a range of Unicode characters. You can specify more than one class for a given element; the values are additive, and in case of conflict the latest value in the CSS file overrides earlier values. The order of values in the `class` attribute itself does not matter. The net effect is that you can use this feature without messing up the display of elements for which you already have other CSS rules. This is essential for the safe use of the feature.

To activate assignment of classes to Unicode character ranges:

```
[CSS]
; UseCharRangeClasses = No (default); or Yes (to activate settings in
; [CharacterRangeClasses] for marking spans by Unicode char range)
UseCharRangeClasses = Yes
```

To specify a class to use for spans of characters:

```
[CharacterRangeClasses]
; starting U+ code point (four or five hex digits) = class name,
; - (exclude from all classes), or * (allow in any class).
xxxx = classname optional comment here
yyyy = * allow in all classes
zzzz = - exclude from all classes
```

The named class applies to the character code specified, plus all following character codes up to the next setting. Any text after the first term (class name or symbol) is a comment. The initial state is `*` (for allow in any class); the last setting should specify `-` (exclude from all classes).

For example, to flag English and European-language text remaining in a Japanese translation:

```
[CharacterRangeClasses]
0021 = latin common symbols
0030 = * digits
003A = latin alpha, some symbols
00A5 = * Yen sign
00A6 = latin Latin-1, diacritics
0342 = greek Greek diacritics
0346 = latin Latin diacritics
0374 = greek Greek letters
03E2 = - Ethiopic and many more
1E00 = latin Latin extended
1F00 = greek Greek extended
2000 = * lots of punctuation
2E80 = - rest of the world
```

To flag Cyrillic in an English document:

```
[CharacterRangeClasses]
0021 = -
0400 = Russian
0514 = -
2000 = *
3000 = -
```

22.7.7 Assigning CSS classes to FrameMaker conditions

You can use CSS to display FrameMaker conditions that you have transferred to HTML attributes, by assigning classes to display the original condition indicators such as color, underline, and strikethrough. See §13.10.3 [Displaying condition indicators in HTML with CSS](#) on page 447.

22.7.8 Using link format names as CSS class names

To automatically use FrameMaker cross-reference format names and hypertext-link character format names as CSS class names in HTML:

```
[CSS]
; XrefFormatIsXrefClass = No (default) or Yes (for xrefs, use the
;   Frame xref format name as the xref class name; for hyperlinks, use
;   the char format name instead. Mainly for DITA @outputclass use.)
;   Default is reversed to Yes if UseCSS=Yes, and for DITA output.
XrefFormatIsXrefClass = Yes
```

When UseCSS=Yes, the default value of XrefFormatIsXrefClass is reversed to Yes; see §22.5 [Understanding how CSS affects other options](#) on page 687.

For DITA XML, the default value of XrefFormatIsXrefClass is Yes; see §15.3 [Specifying general options for DITA](#) on page 483.

22.7.9 Using CSS class names as tags for XML

By default, CSS class names become XML tags in XML output:

```
[CSS]
; ClassIsTag = No (default for HTML/XHTML)
;   or Yes (default for Generic XML)
```

When ClassIsTag=Yes, class names, including those you assign to formats in the [ParaTags] and [CharTags] sections, become XML tags. If ClassIsTag=Yes, also specify [CSS]WriteClassAttributes=No; see §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684.

When ClassIsTag=No, HTML tags and class names are assigned as described in §22.7.2 [Mapping paragraph formats to CSS classes](#) on page 692 and §22.7.3 [Mapping character formats to tags or span classes](#) on page 693.

For example, suppose your FrameMaker document includes catalogued paragraph formats *Chap_Title*, *SubHead*, *Fig*, and *Body*, with the first two assigned HTML tags and the third assigned a class in [ParaTags]. Mif2Go would treat these formats as follows:

<u>FM format</u>	<u>[ParaTags]</u>	<u>ClassIsTag = No</u>	<u>ClassIsTag = Yes</u>
<i>Chap_Title</i>	Chap_Title=H1	<h1 class="chaptitle">	<chaptitle>
<i>SubHead</i>	SubHead=H2	<h2 class="subhead">	<subhead>
<i>Fig</i>	Fig= class="caption"	<p class="caption"	<caption>
<i>Body</i>	(no setting)	<p class="body">	<body>

22.7.10 Omitting tags from CSS selectors

By default, for HTML output Mif2Go writes CSS selectors as class names prefixed with the element tag.

To have Mif2Go write CSS selectors as just class names with no tag prefix:

```
[CSS]
SelectorIncludesTag = Yes (default for HTML output) or No
; (omit element tag prefix, default for DITA and DocBook output)
SelectorIncludesTag = No
```

When SelectorIncludesTag=Yes, CSS selectors consist of the element tag name followed by a period followed by the class name; for example, h1.heading1. This is the default for HTML and XHTML output.

When `SelectorIncludesTag=No`, CSS selectors do not have an element tag as a prefix; for example, `heading1`. This is the default for DITA and DocBook output.

22.7.11 Overriding CSS class for selected paragraphs

Paragraphs that have distinct purposes in your document should have distinct FrameMaker format names, even if they share the same print format. However, if your document does contain paragraphs with the same format name that need different CSS classes, you can use **Code** markers to flag those paragraphs, and assign a different class with a macro.

For example, suppose most of your *Heading 2* paragraphs are assigned CSS class `Heading2`, but a few *Heading 2* paragraphs need one of three other classes: `About`, `Configuration`, or `Procedure`. You can surround all *Heading 2* paragraphs with code to hold the HTML tags and class assignments:

```
[HTMLParaStyles]
Heading 2=NoPara CodeBefore CodeAfter
```

The starting H2 tag assigns a class whose value is computed by macro `$UseH2Class`:

```
[ParaStyleCodeBefore]
Heading 2=<H2 class="<$UseH2Class">">
```

The closing H2 tag follows the paragraph:

```
[ParaStyleCodeAfter]
Heading 2=</H2>
```

Macro `$UseH2Class` checks the value of macro variable `$$h2class` to determine which class to assign:

```
[UseH2Class]
<$_if ($$h2class is "A")>About\
  <$_elseif ($$h2class is "C")>Configuration\
  <$_elseif ($$h2class is "P")>Procedure\
  <$_else>Heading2\
  <$_endif>
<$$h2class="H">\
```

Macro variable `$$h2class` is initialized (and always reset) to a value that results in assigning the default class, `Heading2` (via the `$_else` clause in macro `$UseH2Class`):

```
[MacroVariables]
h2class=H
```

To set `$$h2class` for a paragraph that needs a non-default class, you would insert a **Code** marker in the paragraph that *precedes* each such paragraph. The content of the marker would look like this:

```
<$$h2class="A">
```

To assign a non-default class to the very first paragraph in a FrameMaker file, you would have to create a chapter-specific configuration file, `filename.ini`, for that FrameMaker file, with content (for example):

```
[MacroVariables]
h2class=A
```

See §33.1.1 [Providing configuration files for individual chapters](#) on page 919.

See also:

§28 [Working with macros](#) on page 787

§29 [Working with FrameMaker markers](#) on page 831

§33 [Overriding configuration settings](#) on page 919

22.8 Customizing CSS properties

In this section:

- §22.8.1 [Assigning a CSS generic font family](#) on page 698
- §22.8.2 [Specifying CSS <body> tag properties](#) on page 698
- §22.8.3 [Specifying CSS size values and units of measurement](#) on page 699
- §22.8.4 [Overriding styles in Mif2Go-generated CSS files](#) on page 700
- §22.8.5 [Adjusting leading \(line spacing\) in CSS](#) on page 700
- §22.8.6 [Preventing tags from overriding CSS properties](#) on page 701

22.8.1 Assigning a CSS generic font family

Mif2Go cannot automatically assign the CSS generic-family property to every font used in your document, because there are five possible font families and a huge number of possible fonts. However, you can assign a generic font family to each of the fonts used in your FrameMaker formats. For example:

```
[Fonts]
; Document font name = HTML font name (comma-delimited list allowed)
Arial = arial, helvetica, sans-serif
Century = "new century schoolbook", serif
Courier New = "courier new", courier, monospace
Symbol = symbol, fantasy
```

If an assigned font name contains spaces, surround the name with double quotes.

The generic-family values are serif, sans-serif, monospace, cursive, and fantasy. Specify fantasy for Symbol and WingDing fonts.

22.8.2 Specifying CSS <body> tag properties

You can specify a size value and the unit of measurement for the font-size property of the <body> tag in a **Mif2Go**-generated CSS file. Or, you can direct **Mif2Go** not to include a <body> tag entry in the CSS file; then you can substitute your own entry, in the project configuration file. Use one or the other method to specify a font size other than the default:

[Custom font size and units](#)

[Custom <body> tag entry.](#)

*Custom font size
and units*

To specify font size and unit of measurement for the <body> tag:

```
[CSS]
; CssBodyFontSize = value for body {font-size: }, used as base for all
; em and ex sizes, and for font-size and line-height %, default 10.
CssBodyFontSize=10
; CssBodyFontUnit = units for body {font-size: }, default 0:
; 0=pt, 1=pc, 2=in, 3=cm, 4=mm, 7=px (pixels).
CssBodyFontUnit=0
```

CssBodyFontSize determines how values of relative measures em, ex, and % are computed for other CSS style properties. For example, 1.5em in a style property equals 1.5 times the value in pt (or in another non-relative unit) of the <body> tag font-size property.

CssBodyFontUnit should be an absolute unit of measurement rather than a relative unit.

*Custom <body>
tag entry*

To prevent **Mif2Go** from automatically including a style entry for the <body> tag in a **Mif2Go**-generated CSS file:

```
[CSS]
; CssBodyFontTag = Yes (default, writes body { font-size:} or No
CssBodyFontTag=No
```

To specify the default font size yourself, provide a custom entry in macro section [CSSStartMacro]. For example:

```
[CSSStartMacro]
body { font-size: 11pt; margin: 0 0 0 0 }
```

See also:

§22.8.3 [Specifying CSS size values and units of measurement](#) on page 699

§22.8.4 [Overriding styles in Mif2Go-generated CSS files](#) on page 700

22.8.3 Specifying CSS size values and units of measurement

By default, measurements for properties such as font size and line height are expressed in pt units in **Mif2Go**-generated CSS entries. You can direct **Mif2Go** to use other units instead. For example, if you are generating HTML Help and you want to enable the **Font** button on the toolbar, font sizes are best expressed in em units. Relative units (em, ex, and %) are based on whatever absolute unit (pt, pc, in, cm, mm, or px) is used for the font-size property of the <body> tag entry; see §22.8.2 [Specifying CSS <body> tag properties](#) on page 698.

You can specify how many decimal places **Mif2Go** should use for CSS property values; the default is two decimal places. Trailing zeros in property values are eliminated. For example, if a value is computed to be 1.00em, in the CSS file the value appears as 1em. Fractional values are rounded rather than truncated.

To specify units of measurement for font size and line height in CSS entries:

```
[CSS]
; CssFontUnits = units for font size and line height, default 0:
; 0=pt, 1=pc, 2=in, 3=cm, 4=mm, 5=em, 6=ex (0.5em), 7=px (pixels), 8=%
CssFontUnits=0
; CssFontUnitDec = count of digits to right of decimal in CSS font
; values: 0, 1, or 2, default 2. Trailing zeros are trimmed.
CssFontUnitDec=0
```

To specify units of measurement for paragraph spacing, indentation, and margins:

```
[CSS]
; CssIndentUnits = units for para space and indents, default 0:
; 0=pt, 1=pc, 2=in, 3=cm, 4=mm, 5=em, 6=ex (0.5em), 7=px (pixels), 8=%
CssIndentUnits=0
; CssIndentUnitDec = count of digits to right of decimal in CSS indent
; values: 0, 1, or 2, default 2. Trailing zeros are trimmed.
CssIndentUnitDec=0
; CssIndentBaseSize = value used for computing percents for margin
; settings (para space above and below, and indents) in .css file
CssIndentBaseSize=6
; CssIndentBaseUnit = units for CssIndentBaseSize, default 2 (in)
; 0=pt, 1=pc, 2=in, 3=cm, 4=mm, 7=px (pixels).
CssIndentBaseUnit=2
```

The base unit of measurement for computing margin settings should be an absolute unit, not a relative unit.

See also:

§22.8.2 [Specifying CSS <body> tag properties](#) on page 698

§22.8.4 [Overriding styles in Mif2Go-generated CSS files](#) on page 700

22.8.4 Overriding styles in Mif2Go-generated CSS files

When you direct **Mif2Go** to generate a CSS file anew each time (that is, when `[CSS]WriteCssStylesheet=Always`), styles are updated from the formats in your FrameMaker document, and anything you added directly to the CSS file is lost. However, you can include settings in the configuration file to modify the generated CSS file. With these settings you can do any or all of the following:

- Override CSS code** Replace generated CSS code with fixed CSS code for selected formats.
- Omit CSS code** Prevent CSS code from being written to the CSS file for selected formats.
- Add CSS code** Add code to the beginning or the end of the generated CSS file.

See also:

[§22.8.2 Specifying CSS <body> tag properties](#) on page 698

[§22.8.3 Specifying CSS size values and units of measurement](#) on page 699

Override CSS code To override the style specification in a **Mif2Go**-generated CSS file for a particular FrameMaker format, assign a property to the format, and optionally provide replacement code for the style. For example:

```
[HTMLParaStyles] or [HTMLCharStyles]
; CSSReplace uses [ParaStyleCSS] or [CharStyleCSS] to specify
; on a single line the code to be written to the .css for the
; format when [HtmlOptions]WriteCssStylesheet = Always or Once
; NoCSS suppresses writing info to the .css file for its format.
SomeFmt = CSSReplace
OtherFmt = NoCSS
```

When you assign property `CSSReplace` to a format, you must also specify replacement CSS code for that format in section `[ParaStyleCSS]` for a paragraph format or `[CharStyleCSS]` for a character format. The code assignment must be all on one line. For example:

```
[ParaStyleCSS]
SomeFmt = p.somefmt {font: bold 12pt/14pt sans-serif}
```

Omit CSS code When you assign property `NoCSS` to a format, **Mif2Go** still generates the class attributes in the HTML, but does not include them in the CSS file.

Add CSS code To add starting and ending code to a generated CSS file:

```
[CSSStartMacro]
; CSS code to be inserted at the start of the .css file if generated

[CSSEndMacro]
; CSS code to be inserted at the end of the .css file if generated
```

You can use these macro configuration sections to add more CSS entries, perhaps with selectors **Mif2Go** does not use; or add CSS code for positioning, or to set anchor properties, or `<body>` properties, or special properties for nested items.

22.8.5 Adjusting leading (line spacing) in CSS

By default, **Mif2Go** includes line leading (spacing) information in the CSS file. In some cases, this is not desirable; for example, it messes up printing in Netscape 4.x. You can turn off line leading:

```
[HTMLOptions]
; UseCSSLeading = Yes (default) or No (omit linespacing in CSS files)
UseCSSLeading=No
```

You might have to make some changes to your FrameMaker paragraph formats to get CSS to yield good results, especially if you are trying to get those CSS results out of Netscape. For example, Netscape does a poor job with `margin bottom`, which is equivalent to FrameMaker Space Below, but renders `margin top`, equivalent to Space Above, reasonably well. So you might need to add Space Above to your formats to match the Space Below of the formats they follow, in order to get any inter-paragraph space at all. This might not be as bad as it sounds; if you use consistent spacing between elements, the change might not affect the appearance of your FrameMaker document.

22.8.6 Preventing tags from overriding CSS properties

To keep tags from overriding CSS properties, use the following settings to eliminate the tags entirely; these are the default values when `UseCSS=Yes`:

```
[HtmlOptions]
NoFonts=Yes
Basefont=No
```

See §22.4.2 [Specifying CSS options in a Mif2Go configuration file](#) on page 684.

23 Including graphics in HTML

This section shows which graphic formats to use, and which configuration options to specify, for appropriate image and equation display in HTML, XML, and HTML-based Help. Topics include:

- §23.1 [Starting with default graphics options](#) on page 703
- §23.2 [Understanding graphics processing for HTML](#) on page 703
- §23.3 [Locating graphics files for HTML](#) on page 704
- §23.4 [Specifying options for HTML graphics](#) on page 705
- §23.5 [Selecting and modifying graphics](#) on page 708
- §23.6 [Positioning graphics in HTML output](#) on page 714
- §23.7 [Specifying HTML image attributes](#) on page 718
- §23.8 [Providing \(or omitting\) alternate text for images](#) on page 718
- §23.9 [Scaling images for HTML](#) on page 719
- §23.10 [Creating image maps for HTML](#) on page 722
- §23.11 [Supplying a background image or watermark](#) on page 725
- §23.12 [Converting equations for HTML](#) on page 725

See also:

- §31 [Working with graphics](#) on page 869

23.1 Starting with default graphics options

Try an initial conversion without specifying any graphics options. If you just click one button in the *Export* dialog, **Write for anchored frames**, automatically you get settings that should convert every graphic referenced in your HTML output; see §3.6 [Converting documents](#) on page 82.

The resulting graphic quality (from using FrameMaker graphic export filters) is not always the best possible, but it is good enough for screen-based Help systems. The worst case is for EPS graphics, where you get a conversion of the low-resolution preview image rather than the PostScript image; see §31.2.2.3 [Converting EPS graphics](#) on page 875.

If the results are not satisfactory for one or more graphics, read §23.2 [Understanding graphics processing for HTML](#) on page 703, then decide on additional or alternate graphics options.

Note: Graphics on master pages are not included in HTML output.

23.2 Understanding graphics processing for HTML

When you use **Mif2Go** “out of the box” to generate HTML, without setting graphics options, the graphics from your document might look quite different from the way they look when you “Save as” HTML directly from FrameMaker. Differences can appear in any of the following:

- [Image size](#)
- [Image alignment](#)
- [Image format](#)

Image size Screen captures might be unreadable with “out-of-the-box” options. By default, **Mif2Go** retains the image size specified in your FrameMaker document. However, when you “Save as” HTML, FrameMaker produces the graphics unscaled. To do the same with **Mif2Go**, set the following option:

```
[Graphics]
GraphScale = No
```

This setting determines whether **Mif2Go** writes width and height attributes; when GraphScale=No those attributes are omitted from HTML output.

See §23.9.2 [Adjusting image size for selected graphics](#) on page 720.

Note: When you shrink a screen shot at all, you immediately lose text readability. Shrinking a bitmap means using fewer pixels. Most text has parts that are one pixel wide. When you shrink text, some of those parts disappear entirely. If you do not shrink the image, you get the effect of a “huge” graphic. Your choice. This is a very well known publications problem. It is not a **Mif2Go** problem.

Image alignment By default, **Mif2Go** retains the alignment used in your FrameMaker document. However, when you “Save as” HTML, FrameMaker produces HTML with images all left aligned. To mimic FrameMaker HTML image alignment behavior with **Mif2Go**, set the following option:

```
[GraphAlign]
* = left
```

See §23.6.2 [Aligning anchored graphics](#) on page 714.

Image format Graphics formats that work best in FrameMaker for printed documents generally are not those that work well on the Web. If some of the graphics in your FrameMaker document are not in an appropriate format, or are not alone in their frames, you will have to convert them.

If a graphics file imported into or exported from your document is in a format other than JPEG, GIF, or PNG, and you do not specify how it should be converted or mapped, **Mif2Go** plunks the name of the graphics file into the HTML output file and lets the browser sort it out later. In some cases, this might work. For example, Microsoft Internet Explorer can display BMP and WMF graphics with no problem.

23.3 Locating graphics files for HTML

For standard HTML output to be viewed with a browser, you can place graphics files in the same directory as the HTML files, or in any other directory relative to that directory. For other HTML output types, graphics placement is restricted:

- For HTML Help and OmniHelp, graphics *must* be located either in the same directory as the HTML files, or in a subdirectory at any level below the directory containing the HTML files.
- For JavaHelp and Oracle Help, graphics *must* be located in a subdirectory of the helpset directory, at the same level as the directory containing the HTML files.

Graphics in directory with HTML files If graphics are in the same directory as the HTML files, references to those graphics via tags do not need a path component, and whatever path information is already present in FrameMaker must be removed.

To remove path information from graphics file names:

```
[Graphics]
; StripGraphPath = No (default)
; or Yes (remove path from referenced graphics)
StripGraphPath = Yes
```

When `StripGraphPath=Yes`, **Mif2Go** omits any path information from references in generated `` tags.

Graphics in a different directory

If graphics will be in a directory different from the directory for HTML files, you must specify the path from the HTML files to the graphics directory, so **Mif2Go** can include the path in the generated `` tags.

To specify where a browser (or Help viewer) should look for graphics:

```
[Graphics]
; GraphPath = path to use (replacing any previous) for all graphics
GraphPath = path/to/graphics/files
```

`GraphPath` specifies the location of graphics files relative to the location of HTML files. For Web-hosted systems, `GraphPath` must be the path to the graphics on the *Web server*, which might be different from the file path on the *conversion system*. Although you can specify an absolute path, relative is almost always what you want.

Note: Absolute paths do not work if the graphics are on a UNIX server.

Default path

The default value of `GraphPath` is the directory designated by `[Automation]WrapPath` (see §35.6 [Assembling files for distribution](#) on page 961); if `WrapPath` is not specified, the default is the project directory. For JavaHelp and Oracle Help only, the default value of `GraphPath` is the directory designated by `[JavaHelpOptions]GraphSubdir`, prefixed with “`../`”. See §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

If you do not specify a value for `GraphPath`, the value of `StripGraphPath` determines whether **Mif2Go** includes the original path from your FrameMaker document, or no path at all, in generated `` tags.

GraphPath does not move files!

In HTML references to images, the `GraphPath` setting prefixes the path specified by `GraphPath` to the name of each graphics file, in place of whatever other path was there in your FrameMaker document. This option sets the `src` attribute of the `` tags; *it does not change the location of the graphics files themselves*. You must either copy the graphics files to their specified location, or have **Mif2Go** copy them for you. See §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.

See also:

- §35.7 [Placing graphics files for distribution](#) on page 965
- §9.3.10 [Locating graphics files for HTML Help](#) on page 302
- §10.3.9 [Getting OmniHelp supporting files in the right place](#) on page 349
- §11.3.7.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 380
- §31.3.1.1 [Specifying graphics location for HTML](#) on page 887

23.4 Specifying options for HTML graphics

In this section:

- §23.4.1 [Using referenced graphics without converting](#) on page 706
- §23.4.2 [Specifying formats of replacement graphics](#) on page 706
- §23.4.3 [Choosing a graphics conversion method](#) on page 707
- §23.4.4 [Using referenced, embedded, and compound graphics](#) on page 707
- §23.4.5 [Omitting graphics from HTML or XML output](#) on page 708

23.4.1 Using referenced graphics without converting

If some referenced graphics are already in a format appropriate for Web use, such as JPEG, GIF, or PNG (see §31.1.4 [Graphics formats for HTML](#) on page 871); and if they are alone in their anchored frames (no callouts or in-frame titles, for example); you do not have to convert them. Either check **Use original graphic names** in the *Mif2Go Export* dialog, or set the following option in the configuration file:

```
[Graphics]
UseOriginalGraphicNames=Yes
```

For details, see §31.3.1.4 [Using original files and image sizes for referenced graphics](#) on page 889.

The naming is preserved like any other in the `` tags.

Copy the graphics files into the project directory with the generated .htm files, or allow **Mif2Go** to copy them for you to the wrap directory; see §23.3 [Locating graphics files for HTML](#) on page 704. Unless you explicitly remap a name in the [GraphFiles] section (see §23.4.2 [Specifying formats of replacement graphics](#) on page 706), or specify a `GraphSuffix` in the [Graphics] section, the graphic name is always passed through unchanged.

If the original graphics are not in the same directory as the FrameMaker files that reference them, but they will be in the same directory as the generated HTML files, set the following option also (see §23.3 [Locating graphics files for HTML](#) on page 704):

```
[Graphics]
StripGraphPath=Yes
```

If you have supplied replacements for referenced graphics that are in a different format, and if the replacements have the same base names as the originals, you can specify just the new file extension (see §31.3.1.2 [Substituting graphics files for HTML](#) on page 888):

```
[Graphics]
GraphSuffix=jpg
```

Use this setting when you convert referenced graphics with a third-party program; see §5.7 [Processing graphics](#) on page 126.

A problem arises if you add any FrameMaker elements, such as arrows or callouts, to a referenced graphic. You can direct **Mif2Go** to use the FrameMaker export filters to convert the whole graphic to JPEG or GIF; see §5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129. Or you can reproduce the elements in the external graphic with a third-party program.

23.4.2 Specifying formats of replacement graphics

If you have replaced some referenced graphics with others, and your graphics are in several formats, such as mostly GIF plus some JPEG and some PNG, you can do the following:

1. Identify the “main” format by specifying its file extension; for example:

```
[Graphics]
GraphSuffix=gif
```

2. Specify file extensions for the other formats as exceptions:

```
[GraphSuffix]
; old suffix = new suffix, overrides [Graphics]GraphSuffix
; jpg=jpg   leaves all .jpgs alone even if GraphSuffix=gif
; wmf=png   .wmfs are made into .pngs using a third-party tool
```

```
jpg=jpg
png=png
```

- Specify file names with extensions for any individual exceptions (see §31.3.1.2 [Substituting graphics files for HTML](#) on page 888):

```
[GraphFiles]
newlogo.jpg=newlogo.gif
```

23.4.3 Choosing a graphics conversion method

If some graphics in your FrameMaker document are not already JPEG, GIF, or PNG, use one of the following methods to convert them:

- Let **Mif2Go** use FrameMaker export filters; this method is automatic. By default, **Mif2Go** makes a .jpg file for each graphic, and references them in the `` tags in HTML. For more information, and to fine-tune this process, see the following:
 - §5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129.
 - §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883.
- Use a graphics program; see §5.7.2.3 [Using third-party graphics converters](#) on page 130. If the graphics are embedded in your FrameMaker document, first you will have to export them; see the following:
 - §31.2.3 [Exporting and converting embedded graphics](#) on page 877.
 - §31.3.1.2 [Substituting graphics files for HTML](#) on page 888.

23.4.4 Using referenced, embedded, and compound graphics

Suppose your FrameMaker document includes all of the following:

- [Original referenced graphics](#) (GIFs, for example) that do not need to be converted
- [Embedded graphics](#) that must be exported
- [Compound graphics](#): images in anchored frames that include callouts or other drawing elements created in FrameMaker.

*Original
referenced
graphics*

Mif2Go undertakes two distinct operations:

- Mif2Go** generates graphics files for all anchored frames. This is an all-or-nothing operation, which is required if a document contains any graphics with callouts.
- Mif2Go** writes the HTML files. This operation does not have to use the graphics produced in Step 1.

To have **Mif2Go** use the original referenced graphics—*except* for those with callouts—you would set the following option:

```
[Graphics]
UseOriginalGraphicNames=Yes
```

Whenever you have a referenced graphic, and it is alone in its frame (no callouts, title, second image, and so on, in the frame), the resulting HTML uses that graphic. For all other cases the resulting HTML uses the graphics **Mif2Go** generates from your document.

*Embedded
graphics*

To have **Mif2Go** export embedded graphics (those without callouts) to files of their own, you would set the following option:

```
[GraphExport]
ImportGraphics=Export
```

The exported files are referenced in the resulting HTML just as though they had been referenced in FrameMaker in the first place. See §31.2.3 [Exporting and converting](#)

[embedded graphics](#) on page 877 for information about the appropriate export settings to use.

*Compound
graphics*

Graphics that include callouts or other elements created in FrameMaker must be processed with FrameMaker export filters; see §5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129. The only other option would be to use a third-party graphics program to add the callouts to the original graphic. If any compound graphics include images imported into FrameMaker at a resolution other than 96 DPI, you would need to scale the graphics on export; see §23.9 [Scaling images for HTML](#) on page 719.

23.4.5 Omitting graphics from HTML or XML output

To strip all graphics from your document so no tags or references to graphics are included in HTML or XML code, substitute for the graphics a macro that does nothing:

```
[GraphReplaceMacros]
* = <$nothing = 1>
```

The macro must have some content; a simple variable assignment produces no output. You might also need the following settings to avoid having **Mif2Go** waste time generating graphics in the first place, and to eliminate any anchor paragraphs for the graphics:

```
[Graphics]
UseGraphicPreviews = No
GraphWrapPara = No
```

See also:

§3.7.4.1 [Omitting and restoring graphics production](#) on page 86

§23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710

§23.5.6 [Omitting paragraph tags around graphics](#) on page 713

23.5 Selecting and modifying graphics

In this section:

§23.5.1 [Assigning properties to sets of graphics](#) on page 708

§23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710

§23.5.3 [Converting only the visible portion of a graphic](#) on page 712

§23.5.4 [Converting reference-page graphics for HTML](#) on page 712

§23.5.5 [Eliminating graphics in unanchored frames](#) on page 713

§23.5.6 [Omitting paragraph tags around graphics](#) on page 713

§23.5.7 [Retaining run-in images in otherwise empty paragraphs](#) on page 713

23.5.1 Assigning properties to sets of graphics

You can assign properties to, and override default configuration settings for, both individual graphics and selected groups of graphics.

In this section:

§23.5.1.1 [Using wildcards to assign properties to graphics](#) on page 709

§23.5.1.2 [Using markers to assign properties to graphics](#) on page 709

§23.5.1.3 [Specifying an image class for a graphic](#) on page 710

§23.5.1.4 [Creating named groups of graphics](#) on page 710

See also:

§5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117.

23.5.1.1 Using wildcards to assign properties to graphics

To apply a setting to a subset of all the graphics in your document, you can use ? or * wildcards in GraphicIDs. To exclude a graphic, assign nothing to its GraphicID. For example, to selectively scale images:

```
[GraphScale]
; Do not scale the following images:
aa568433=
ab00b5d3=
; Scale the following image to 75% of its original size:
ab123456=75
; Scale all other images in the chapter to 50%:
ab*=50
; Scale all remaining images in the book to 25%:
*=25
```

See also:

§4.6 [Using wildcards in configuration settings](#) on page 106.

§5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117.

23.5.1.2 Using markers to assign properties to graphics

You can use markers in your FrameMaker document to assign a property to a single graphic or to only a few graphics, or to exclude a graphic from a general assignment. In some cases this might be easier than determining the individual GraphicIDs required in configuration settings. Use either of these marker types:

HTMConfig Content is `[GraphSection]=Value`

HTML Macro Content is any HTML code

HTMConfig for individual graphics

Insert the **HTMConfig** marker in text before the graphic, and provide as marker content the property assignment. For example, to scale a certain graphic to 75%, you could place an **HTMConfig** marker just before the anchor for the graphic frame, and specify the scale factor as the marker content:

```
[GraphScale]=75
```

See §33.2.9.4 [Overriding graphic properties for HTML](#) on page 929.

HTML Macro for a series of graphics

You can use markers of type **HTML Macro** to change the value of a macro variable just before a graphic or series of graphics, then change it back again after the graphics. For example, you could use **HTML Macro** markers and a macro variable to scale a series of graphics to 75%.

Include in the configuration file a scale-factor setting that references a macro variable:

```
[GraphScale]
*=<$$scalepct>
```

Initialize the value of the macro variable:

```
[MacroVariables]
scalepct=100
```

In text just before the graphics to be scaled, insert an **HTML Macro** marker with content:

```
<$$scalepct=75>
```

Just after the graphics to be scaled, insert another **HTML Macro** marker with content:

```
<$$scalepct=100>
```

See §28.3 [Using macro variables](#) on page 795.

23.5.1.3 Specifying an image class for a graphic

Mif2Go provides two ways to assign a CSS class to the tag created for a graphic:

[Attribute marker](#)

[Object property](#).

Attribute marker Insert a **GraphClass** marker in text preceding the graphic. Make the content of the marker the name of the image class. See §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.

Object property Use the FrameMaker *Object Attributes* dialog to specify an image class. See §31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896.

Mif2Go does not provide a way to assign an image class via paragraph format, because generally authors use the same anchor paragraph for all types of graphics.

See also:

§23.7 [Specifying HTML image attributes](#) on page 718

23.5.1.4 Creating named groups of graphics

To apply the same settings to several graphics, you can create a *graphics group* by assigning a common group name to the GraphicIDs of the graphics in question. For example:

```
[GraphGroup]
; Graphic ID = graphic group name, any name you want
ab01f853=schematic
ab012c13=schematic
aa568433=screenshot
ab00b5d3=screenshot
```

Once you have assigned a group name to one or more graphics, in any of the other [Graph*] sections you can assign properties to the group name instead of to a GraphicID. The values you assign affect all graphics defined as belonging to the named group, except any graphics to which you explicitly assign a different value.

For example, to avoid scaling screenshots, but reduce schematics in size:

```
[GraphScale]
; Do not scale images in the screenshot group:
screenshot=
; Scale schematics to 25% of their original size:
schematic=25
```

Another way to assign a graphic to a group is to insert an **HTMConfig** marker in text just before the graphic in your FrameMaker document, with content as follows:

```
[GraphGroup]=graphicgroupname
```

This way you can avoid having to look up GraphicIDs and FileIDs. See §33.2.9.4 [Overriding graphic properties for HTML](#) on page 929.

23.5.2 Replacing or surrounding a graphic with macro code

You can specify code to be included before, after, or in place of any graphic, or group of graphics, by assigning a macro or HTML code to one or more graphic IDs. For example:

```
[GraphStartMacros]
; Graphic ID = text of macro to put before graphic
ax78ec24=<hr /><br />
```

```
[GraphEndMacros]
; Graphic ID = text of macro to put after graphic
ax78ec24=<hr />

[GraphReplaceMacros]
; Graphic ID = text of macro to put instead of graphic
aq*=<$Thumbnail>
```

When you specify a macro or other HTML code to *replace* a graphic, **Mif2Go** ignores any preceding or following code or macro you assign to that same graphic in one of the *other* [Graph*Macros] sections.

To avoid having graphics wrapped in paragraph tags when you use [GraphReplaceMacros], see §23.5.6 [Omitting paragraph tags around graphics](#) on page 713.

List exceptions by graphic ID

If the macro should apply to all but a few images, you can list the images to exclude by assigning nothing to their IDs; list the exceptions first. For example:

```
[GraphReplaceMacros]
aa12345=
aa23456=
*=<$YourMacro>
```

See §23.5.1.1 [Using wildcards to assign properties to graphics](#) on page 709.

Use predefined macro variables

The macro definition (or HTML code) can include the following predefined macro variables, which reference the graphics to which the code or macro is assigned:

<\$\$_graphbase>	File name for attribute, without extension
<\$\$_graphsrc>	File name for attribute, with extension
<\$\$_graphorighigh>	Original image height in pixels, before any [GraphScale], [GraphHigh], or [GraphWide] setting is applied
<\$\$_graphorigwide>	Original image width in pixels, before any [GraphScale], [GraphHigh], or [GraphWide] setting is applied

If you assign code instead of a macro name, the code must be all on the same line.

Replace graphics with thumbnails

To show each graphic in a smaller size (a “thumbnail”), for example, you could specify something like the following (see §4.6 [Using wildcards in configuration settings](#) on page 106):

```
[GraphReplaceMacros]
*=<$Thumbnail>

[Thumbnail]
<a href="<$$_graphsrc>"></a>
```

For a way to provide thumbnails in the form of links to the graphics they replace, see §18.7.3.2 [Using thumbnails for links to illustrations in HTML](#) on page 604.

View full-size graphics on demand

If you import high-resolution bitmap images into your document by reference, and in FrameMaker you scale them down to fit the page, the scaled-down images might not show clearly in HTML. You can make these images clickable in HTML, so the graphic can be viewed full size. For example:

```
[GraphReplaceMacros]
aa4de33f=<$FullView>

[FullView]
<a href="<$$_graphsrc>" target="_blank">" height="<$$_graphorighigh>" /></a>
```

23.5.3 Converting only the visible portion of a graphic

If your FrameMaker document includes referenced graphics that do not require conversion, you would most likely specify `UseOriginalGraphicNames=Yes` (see §23.4.1 [Using referenced graphics without converting](#) on page 706). However, suppose that for some of these graphics, the anchored frame purposely shows only a portion of the image. To use only the portion of an image visible in FrameMaker, such a graphic must be converted. You have two choices:

- Add an empty Text Line to each graphic in FrameMaker. This element would be invisible in FrameMaker and in all outputs, but would force **Mif2Go** to use FrameMaker export filters to convert the graphic.
- Include configuration macros to surround each graphic with macros that turn off `UseOriginalGraphicNames` and then turn it back on again.

To use configuration macros, assign configuration variables to the **Mif2Go** graphic ID (see §5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117) for each graphic:

```
[GraphStartMacros]
graphicframeID = <$$[Graphics]UseOriginalGraphicNames=0>

[GraphEndMacros]
graphicframeID = <$$[Graphics]UseOriginalGraphicNames=1>
```

These settings would cause **Mif2Go** to use FrameMaker export filters to convert just the portion of `graphicframeID` that shows within its anchored frame.

See also:

§23.4.1 [Using referenced graphics without converting](#) on page 706

§23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710

§33.2.3 [Overriding settings with macros](#) on page 921

§33.2.9.4 [Overriding graphic properties for HTML](#) on page 929

23.5.4 Converting reference-page graphics for HTML

Mif2Go uses FrameMaker export filters to convert reference-page graphics created with FrameMaker drawing tools; see §31.2.5.7 [Converting graphics on reference pages](#) on page 885. You might or might not want to include the converted graphics in HTML output; see §21.3.7 [Keeping or removing reference frames](#) on page 651.

In this section:

§23.5.4.1 [Replacing reference-frame horizontal rules](#) on page 712

§23.5.4.2 [Suppressing indentation of reference-page graphics](#) on page 713

23.5.4.1 Replacing reference-frame horizontal rules

For graphics such as the `FrameAbove` and `FrameBelow` horizontal lines used around some FrameMaker formats, you might get better results by replacing the reference-page graphics with HTML horizontal rules. This approach avoids carrying an extra generated .jpg around for every FrameMaker file you convert. For example:

```
[HTMLParaStyles]
Note = CodeBefore CodeAfter NoFrameAbove NoFrameBelow

[ParaStyleCodeBefore]
Note = <br /><hr />

[ParaStyleCodeAfter]
Note = <hr /><br />
```

You can tune the `<hr>` rules by specifying width, height, or other attributes, and possibly by using a class to apply CSS properties; see §22.7.2 [Mapping paragraph formats to CSS classes](#) on page 692.

23.5.4.2 Suppressing indentation of reference-page graphics

By default, **Mif2Go** uses `1p.gif` to indent graphics; see §23.6.3 [Indenting images](#) on page 716. To prevent **Mif2Go** from indenting reference-page graphics:

```
[Graphics]
; RefPageGraphIndent = Yes (treat reference-page graphic indents
; normally) or No (do not indent)
RefPageGraphIndent=No
```

When `RefPageGraphIndent=No`, **Mif2Go** does not put any spacers before graphics converted from FrameMaker reference pages.

See also:

§21.3.7 [Keeping or removing reference frames](#) on page 651

§23.6.3 [Indenting images](#) on page 716

§31.2.5.7 [Converting graphics on reference pages](#) on page 885

23.5.5 Eliminating graphics in unanchored frames

Unanchored frames do not often occur in properly constructed FrameMaker documents, except on master pages. When **Mif2Go** encounters an unanchored frame on a body page, by default **Mif2Go** anchors the frame to the first paragraph on that page. This might not produce the effect you want. Also, an empty unanchored frame on a body page results in a “missing” image for a mystery graphic that does not appear in the FrameMaker document.

To exclude unanchored frames on body pages from HTML output, specify the following setting:

```
[HTMLOptions]
; ReAnchorFrames = Yes (default, anchor unanchored frames to first
; para on page) or No (skip unanchored frames)
ReAnchorFrames = No
```

23.5.6 Omitting paragraph tags around graphics

By default, **Mif2Go** wraps each non-inline graphic in paragraph tags. If you are replacing graphics with macro code (see §23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710) or repositioning graphics in HTML or XML output, you might need to eliminate the enclosing paragraph tags.

To omit paragraph tags around graphics:

```
[Graphics]
; GraphWrapPara = Yes (default, wrap graphics that are not inline in
; paragraph tags) or No (eliminate wrapping tags)
GraphWrapPara = No
```

23.5.7 Retaining run-in images in otherwise empty paragraphs

If you specify that empty paragraphs should be omitted from output, either via `RemoveEmptyParagraphs` or via `RemoveEmptyTableParagraphs`, you can retain any run-in images in those paragraphs:

```
[Graphics]
; RetainRuninImagesForEmptyParagraphs = No (default) or Yes
RetainRuninImagesForEmptyParagraphs=Yes
```

When `RetainRuninImagesForEmptyParagraphs=Yes`, for an otherwise-to-be-omitted empty paragraph that includes an image set to *Run into Paragraph*, the image is included in output even if the paragraph tags are omitted. **Mif2Go** writes run-in images after any frames that come before the paragraph, and before any frames that come after the paragraph. The run-in images precede any content you specify for the empty paragraph.

See also:

§21.3.10 [Eliminating empty paragraphs in text](#) on page 652

§24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744

23.6 Positioning graphics in HTML output

Mif2Go might not be able to reproduce in HTML the exact position of a graphic in FrameMaker, because HTML does not support the same kind of positioning; it cannot, because HTML has to adjust to variable page sizes. However, you can use **Mif2Go** configuration settings to specify certain alignment options.

HTML positioning attributes are not compatible with CSS; therefore, by default, **Mif2Go** omits these attributes for HTML output.

In this section:

§23.6.1 [Positioning graphics anchored in empty paragraphs](#) on page 714

§23.6.2 [Aligning anchored graphics](#) on page 714

§23.6.3 [Indenting images](#) on page 716

§23.6.4 [Adding space above an image](#) on page 717

§23.6.5 [Eliminating space above or below graphics in table cells](#) on page 717

See also:

§23.5.6 [Omitting paragraph tags around graphics](#) on page 713

23.6.1 Positioning graphics anchored in empty paragraphs

In HTML, unlike in FrameMaker, a graphic need not be anchored in a paragraph. If your FrameMaker document contains graphics anchored At Insertion Point in their own otherwise empty paragraphs, to indent these graphics correctly **Mif2Go** might place them *before* the paragraph that anchored them.

To prevent **Mif2Go** from moving such graphics out of their anchor paragraphs:

```
[Graphics]
; KeepGraphicsInPara = No (default)
; or Yes (keep graphics in para)
KeepGraphicsInPara = Yes
```

23.6.2 Aligning anchored graphics

To override the FrameMaker alignment of an individual graphic or a group of graphics (for example):

```
[GraphAlign]
; Graphic ID = desired alignment to text, one of these: left, right,
; top, texttop, middle, absmiddle, baseline, bottom, or absbottom
ImgGroupA = left
ch01f853 = absmiddle
```

The [GraphAlign] section sets the HTML align attribute of the tag itself; this attribute controls only vertical position and left/right floats in HTML.

*Floating graphics
might hide text*

If you set alignment to left or right, you are also telling the graphic to “float”. This might result in the text that follows (such as a caption, if captions are below the image in your FrameMaker document) disappearing behind the image. If this happens, for the graphic involved you should also specify:

```
[GraphEndMacros]
* = <br clear="all" />
```

Or, you can specify "left" or "right" instead of "all", depending on the effect you want.

*Realign graphics
independently of
anchors*

To position graphics independently of the paragraphs in which their frames are anchored, if you are using CSS you must also specify the following:

```
[Graphics]
; GraphAlignAttributes = Yes (default, allow when set in [GraphAlign])
; or No (no align attribute in img tags even if set in [GraphAlign]).
; Default is reversed to No if UseCSS=Yes.
GraphAlignAttributes = Yes
```

When you use CSS, by default **Mif2Go** ignores any alignment attributes you specify in [GraphAlign]; see §22.5 [Understanding how CSS affects other options](#) on page 687. If you are not using CSS, by default **Mif2Go** uses the alignment attributes in [GraphAlign].

*Realign graphics
horizontally*

For graphics in anchored frames that are neither inline nor run-in (float in HTML) you can specify a horizontal position different from that used in FrameMaker:

```
[GraphParaAlign]
; Graphic ID = desired alignment for containing para: left, right,
; or center, primary method of centering standalone graphics
ch01f853 = left
```

The [GraphParaAlign] property sets the HTML align attribute for the paragraph in which the graphic frame is anchored; this is how **Mif2Go** controls horizontal alignment of graphics. For example, the following setting uses a wildcard (see §4.6 [Using wildcards in configuration settings](#) on page 106) to center-align all anchored graphics in HTML:

```
[GraphParaAlign]
* = center
```

However, the align="center" attribute does not work as specified by the W3C in most browsers.

*Use CSS to
center graphics*

When you use CSS, you can center graphics with **Mif2Go** macros. For example, to center all images:

```
[GraphStartMacros]
* = <div class="img">

[GraphEndMacros]
* = </div>
```

See §23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710.

Include in CSS:

```
div.img {text-align: center; }
```

Or, if **Mif2Go** maintains CSS for your project (see §22.8.4 [Overriding styles in Mif2Go-generated CSS files](#) on page 700):

```
[CSSEndMacro]
div.img {text-align: center; }
```


You have to use `<div>` because CSS applies `text-align` only to block elements, and `` is not a block element.

23.6.3 Indenting images

Best practice is to use CSS to indent images in HTML. The technique described in this section should be used only if you cannot use CSS.

Mif2Go can indent all non-inline graphics to match the indent used in your FrameMaker document. Graphics (and also tables) are indented in HTML output using a technique invented by Chuck Musciano

<http://www.drdobbs.com/184411862>

This technique consists of placing, at the start of the line that contains the graphic, a one-pixel transparent GIF image, `lp.gif`, with a width attribute that produces the required indent in pixels. See §28.2.2 [Modifying Mif2Go-supplied macro definitions](#) on page 793.

To use the built-in **Mif2Go** spacer graphic:

```
[HTMLOptions]
; UseSpacers = No (default)
; or Yes, use to position tables and graphics
UseSpacers = Yes
```

To use your own graphic as a spacer instead of the built-in graphic:

```
[HTMLOptions]
; WriteSpacerFile = No (default) or Yes, write file after conversion
WriteSpacerFile = Yes
```

When `WriteSpacerFile=Yes`, the default name of the indent spacer image file is `lp.gif`; you can specify a different name, or specify a different path:

```
[HTMLOptions]
; PixelSpacerImage = name of 1-pixel transparent GIF for spacing
PixelSpacerImage = lp.gif
```

By default, **Mif2Go** writes the spacer image file to the project directory, and includes references of the following form in your HTML output:

```

```

If you supply a path for `PixelSpacerImage` (for example, `./graphics/lp.gif`), **Mif2Go** writes the spacer image file to the specified path. Then **Mif2Go** generates references of the form:

```

```

This can be important if you place graphics anywhere but the project directory (see §23.3 [Locating graphics files for HTML](#) on page 704).

Spacer alt attribute

The spacer graphic must have an `alt` attribute for W3C validation. The default value for the spacer `alt` attribute is `[spacer]`; you can change this default:

```
[HTMLOptions]
; SpacerAlt = text to use for alt attribute for spacer,
; default [spacer]
SpacerAlt = [spacer]
```

Left indent

You can specify a custom indent for a single graphic or a graphics group; for example:

```
[GraphIndents]
; Graphic ID = number of pixels to indent using PixelSpacerImage
; zero prevents indent, -1 is autoindent (default action)
schematic = 30
```

See §23.5.1 [Assigning properties to sets of graphics](#) on page 708.

Right indent A similar method creates a space to the right of the image, except that the `height` attribute of the spacer is set to match the `height` attribute of the image. This is useful for run-in graphics, and for other floating types. For example:

```
[GraphRightSpacers]
; Graphic ID = number of pixels space on right using PixelSpacerImage
ch00b5d3 = 45
```

No indent If you do not want any graphics indented, use a wildcard setting, as follows:

```
[GraphIndents]
* = 0
```

See §4.6 [Using wildcards in configuration settings](#) on page 106.

23.6.4 Adding space above an image

To add a space above an image, assign an HTML `
` tag to one of the following:

- the format of the anchor-containing paragraph (if it is specific to graphics)
- the ID of the graphic, which consists of the **Mif2Go** FileID followed by the FrameMaker ID; see §5.3 [Identifying files and objects](#) on page 117 for more information.

If all the graphics in your document need the added space, dedicate a paragraph format to anchoring graphics, and include the extra space in its definition. If you need to add space to only one or a few graphics, use the Graphic ID; see §23.5.1 [Assigning properties to sets of graphics](#) on page 708.

Use the anchor paragraph

To add space using the paragraph containing the anchor (for example, *GraphAnchor*):

```
[HTMLParaStyles]
GraphAnchor = CodeBefore

[ParaStyleCodeBefore]
GraphAnchor = <br>
```

This method adds space for every graphic whose frame is anchored in a *GraphAnchor* paragraph.

Use the Graphic ID

To add space using the Graphic ID:

1. **Determine the FrameMaker object ID.** Click the anchored frame; FrameMaker displays the ID on the status bar at the bottom of the window; for example, 0f9fae.
2. **Determine the FileID.** Look in file `mif2go.ini` (located in the same directory as your FrameMaker document) for the name of the file containing the graphic. Under `[FileID]` you will see a list of two-letter codes, each assigned to one of the files in your document; for example, `aa=chap1`.
3. **Add a setting for the graphic.** For example:

```
[GraphStartMacros]
GraphicID = <br>
```

This method produces a space just before the `` tag.

See also:

§23.9.1 [Excluding image size attributes from HTML](#) on page 720

23.6.5 Eliminating space above or below graphics in table cells

When you place an image in an anchored frame inside a table cell, properties of the anchor paragraph can cause unwanted space to appear above or below the image.

To eliminate spacing caused by the anchor paragraph, give that paragraph a special format; for example, *CellPic*; and assign *CellPic* the following properties:

```
[HTMLParaStyles]
CellPic = NoPara NoTags
```

See §21.3.6 [Stripping paragraph properties](#) on page 650.

23.7 Specifying HTML image attributes

You can specify attributes for the `` tag in any of the following ways:

[Configuration-file settings](#)

[Custom markers in FrameMaker](#)

[FrameMaker object attributes.](#)

Configuration-file settings

To specify `` tag attributes in the configuration file (for example):

```
[GraphAttr]
; Graphic file name (with or without extension) = desired attributes
ch01f853.gif= usemap="#schematic" border="0"
```

To eliminate anchored-frame borders from all graphics:

```
[GraphAttr]
*= border="0"
```

Custom markers in FrameMaker

You can define a custom marker that has a name beginning with **Graph** and ending with the name of a valid `` tag attribute (see §29.2.4 [Using attribute markers for HTML or XML](#) on page 835). The content of the marker becomes the value of the attribute for the next image in your document. The marker overrides any configuration-file setting for the same attribute for that image. See §33.2 [Overriding settings with markers or macros](#) on page 920.

FrameMaker object attributes

For images in anchored frames, in FrameMaker 7.0 and later versions you can assign attributes via the *Object Attributes* dialog, shown in [Figure 31-1](#) on page 898. Select an anchored frame and choose **Object Properties...** from the right-click context menu or the FrameMaker **Graphics** menu. In the *Object Properties* dialog, click **Object Attributes...** to open the *Object Attributes* dialog. See §31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896.

See also:

§23.9 [Scaling images for HTML](#) on page 719

§25.2 [Applying WAI markup to images](#) on page 756

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

23.8 Providing (or omitting) alternate text for images

Most current browsers display the content of the `alt` attribute of an `` tag only when the image itself is not displayed.

Note: To show a text value in a tooltip when you move the pointer over an image, use the `title` attribute instead.

By default, if you do *not* provide `alt` text for an image, **Mif2Go** includes an empty `alt` value, to satisfy validators. To omit empty `alt` attribute values:

```
[Graphics]
; AllowEmptyAlt = Yes (default) or No, omit empty alt attributes.
AllowEmptyAlt = No
```

You can specify a value for the `` tag `alt` attribute in any of the following ways:

Include the text in a paragraph
 Insert the text with a marker
 Add the text via Object Attributes
 Assign the text to the graphic file.

*Include the text in
a paragraph*

To use a dedicated paragraph format for alternate text, place a paragraph containing the text in your FrameMaker document, just before the image, and assign the following properties to the paragraph format:

```
[HTMLParaStyles]
AltParaFmt = Alt Delete
```

The `Delete` property prevents the alternate text from appearing as part of the HTML output. See §25.2.2 [Assigning WAI image attributes with dedicated formats](#) on page 757.

To use a character format instead of a paragraph format to provide a value for the `alt` attribute, assign it in `[HTMLCharStyles]` instead of in `[HTMLParaStyles]`.

*Insert the text
with a marker*

To provide alternate text with a marker, insert a marker of type **GraphAlt** in your FrameMaker document, just before the image. The content of the **GraphAlt** marker becomes the value of the `alt` attribute for the next image. See §25.2.3 [Assigning WAI image attributes with custom markers](#) on page 757 and §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.

*Add the text via
Object Attributes*

For images in anchored frames, in FrameMaker 7.0 and later versions you can provide a value for the `alt` attribute via the *Object Attributes* dialog. See §23.7 [Specifying HTML image attributes](#) on page 718 and §31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896.

*Assign the text to
the graphic file*

To assign alternate text to the graphic file (for example):

```
[GraphALT]
; Graphic file name (with or without extension) = desired alt text
ch01f853.gif = Schematic of tuner
```

This method is *not recommended* if your document includes image maps, and you also use the `[GraphALT]` section to assign alternate text to hotspot `<area>` tags. You could lose the `alt` content you assign to the `` tags. See §23.10.2 [Providing alternate text for a hotspot in an image map](#) on page 723.

23.9 Scaling images for HTML

Mif2Go calculates image width and height attributes based on the size of the anchored frame, and on the resolution at which a referenced graphic was imported into FrameMaker. You can override or eliminate the size attributes, and adjust the resolution of exported images.

In this section:

- §23.9.1 [Excluding image size attributes from HTML](#) on page 720
- §23.9.2 [Adjusting image size for selected graphics](#) on page 720
- §23.9.3 [Adjusting image resolution for referenced graphics](#) on page 721
- §23.9.4 [Specifying image resolution for exported graphics](#) on page 721
- §23.9.5 [Specifying px units for graphics sized in pixels](#) on page 722

23.9.1 Excluding image size attributes from HTML

By default, **Mif2Go** includes image width and height attributes in HTML, XHTML, DITA XML, and DocBook XML output, and excludes these attributes from generic XML output.

To exclude image width and height attributes from HTML output:

```
[Graphics]
; GraphScale = Yes (default) to put out width and height attributes,
; or No to eliminate them all (mainly for Generic XML)
GraphScale = No
```

If you do not include any setting at all for GraphScale, you get the default for the output type you specify.

Note: You get faster display in browsers if you keep the image width and height attributes, because a browser can proceed with page layout without waiting for the graphic file to arrive.

See also:

§14.4.3 [Eliminating HTML attributes and tags for generic XML](#) on page 463

§15.7.3 [Omitting size attributes from images for DITA output](#) on page 518

§17.7.3 [Omitting size attributes from images for DocBook](#) on page 582

23.9.2 Adjusting image size for selected graphics

Mif2Go calculates pixel height and width based on the FrameMaker dimensions of each image, at 96 DPI, which is the Windows standard. If necessary you can adjust the size of an image to do any of the following:

[Preserve aspect ratio](#)

[Suppress scaling](#)

[Specify width and height separately.](#)

Preserve aspect ratio To override both width and height of selected graphics, preserving the aspect ratio of each image; for example, to 75% of the original size:

```
[GraphScale]
; Graphic ID = percent of original size to scale (both dimensions)
GraphID = 75
```

This setting affects HTML attributes directly, whether or not you use FrameMaker export filters to generate the graphics.

Suppress scaling To suppress scaling for selected graphics:

```
[GraphScale]
GraphID = 0
```

Setting the percent to zero suppresses scaling because **Mif2Go** does not write width and height attributes that have zero values.

Specify width and height separately To override width and height separately for selected graphics, whether or not you use

```
[GraphScale]:
[GraphWide]
; Graphic ID = number of pixels wide, 0 to omit width attribute

[GraphHigh]
; Graphic ID = number of pixels high, 0 to omit height attribute
```

However, be aware of the following issue with hard-coding the sizes of the graphics you reference in HTML files: localized graphics sometimes have a different size, and hard-coded sizes cause distortion.

23.9.3 Adjusting image resolution for referenced graphics

To adjust for images imported into FrameMaker at a DPI other than 96 (for example, 100):

```
[HTMLOptions]
; ConversionDPI = 96 (default), used when converting sizes to pixels
ConversionDPI = 100
```

This setting adjusts all graphic-related dimensions, including indents, after other scaling factors are applied. It does not affect graphic generation.

For graphics imported by reference, **Mif2Go** graphics “processing” consists of leaving the image alone, and using HTML size settings to achieve the DPI you specify. This means that if you rescale from the original size, you are relying on browser scaling, which is usually (but not always) better than what the FrameMaker export filters would give you.

Resolution will always be poor if you display at any size other than the original. For example, the text in a screenshot has many lines that are just one pixel thick. If you reduce the size, some of those pixels show up as a pixel in the output, and some do not. There is no way around this, and the result is unreadable. For print, you get away with this because a printer renders images at 300 DPI or better, often much better. That is, the printer uses smaller pixels than the screen, so you can shrink the image just fine. But your screen is always at 96 DPI in Windows. So when you display a screenshot, it must take up the same size on screen that it did originally, or it will look awful. You cannot make the pixels any smaller.

Two possible remedies for images that are too large:

- Crop the images so that only the part of interest is shown.
- Substitute thumbnails of the images; clicking a thumbnail opens the full image in a separate window. See §23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710.

23.9.4 Specifying image resolution for exported graphics

When you direct **Mif2Go** to use FrameMaker export filters to generate graphics files from the illustrations in your document, you can specify the DPI of those graphics:

```
[Setup]
; GraphicExportDPI = number (from 50 to 1200, default 96)
```

For compound graphics that include referenced images, specify the same DPI used to import the referenced images into FrameMaker. For example, if images were imported into FrameMaker at 144 DPI, set the export option accordingly:

```
[Setup]
GraphicExportDPI = 144
```

To override the `GraphicExportDPI` value for individual graphics (or groups of graphics):

```
[GraphDPI]
; Graphic ID = DPI to use when generating,
; overrides [Setup]GraphicExportDPI
```

This is something you are not likely to want to do routinely. The default DPI value (96, or whatever was used to import any images involved) usually is best. Changing the

GraphicExportDPI setting (or changing an individual [GraphDPI] setting) affects a displayed HTML page only if browser scaling is turned off for the images involved.

An alternative is to define a custom marker named **GraphDpi** (see §29.2 [Adding custom marker types](#) on page 832). Insert a **GraphDPI** marker, whose content is the DPI value you want, anywhere before the anchor of the anchored frame containing the graphic; the marker applies to the next anchor in the flow. This marker overrides any configuration-file DPI setting for that graphic.

See also:

§23.9.3 [Adjusting image resolution for referenced graphics](#) on page 721

§31.2.5.5 [Specifying graphic output format and DPI](#) on page 884.

23.9.5 Specifying px units for graphics sized in pixels

By default, for all HTML and XML outputs except JavaHelp, **Mif2Go** adds a px suffix to width and height attribute values for images sized in pixels. For example:

```

```

However, a px suffix causes the JavaHelp viewer to show an image as a thumbnail; so for JavaHelp, the default is to omit the suffix. You can direct **Mif2Go** to omit the px suffix for other output types.

To omit the px suffix from image width and height attribute values:

```
[Graphics]
; UsePxSuffix = Yes (default except for JavaHelp, include "px" in the
; width and height attributes), or No (JavaHelp default)
UsePxSuffix = No
```

For DITA XML output, it is usually best to include the px suffix; however, see §15.7.7 [Understanding why images might look incorrectly scaled](#) on page 519.

23.10 Creating image maps for HTML

FrameMaker can place invisible hotspot areas over a graphic, so that clicking different parts of the graphic causes hypertext jumps to different locations. **Mif2Go** automatically converts such graphics into corresponding HTML client-side image maps.

In this section:

§23.10.1 [Creating hotspots for image maps](#) on page 722

§23.10.2 [Providing alternate text for a hotspot in an image map](#) on page 723

§23.10.3 [Specifying jumps from image maps in framesets](#) on page 725

23.10.1 Creating hotspots for image maps

You can provide a single hotspot or multiple hotspots per image map:

[Multiple hotspots per image](#)

[Single hotspot per image.](#)

*Multiple hotspots
per image*

To create an image map with multiple hotspots:

1. Prepare the graphic as you normally do, placing it in an anchored frame.
2. Place a text frame *inside* the anchored frame, wherever you want a hotspot; you can expand the text frame to cover whatever part of the graphic you want included in the hotspot.

- Click inside the text frame, and insert a hypertext **gotolink** marker. Or you can insert a **message URL** hypertext link, to create a jump to a destination outside the FrameMaker document. See §34.1.2 [Using markers to add links and instructions](#) on page 935.

Single hotspot per image

To create an image map with a single hotspot that includes the entire image:

- Place an anchored frame At Insertion Point in an empty paragraph.
- Place the graphic in the anchored frame, and shrink-wrap it.
- Put the hypertext link marker in the same paragraph.
- To avoid having the placeholder paragraph itself show in the output, include the following setting in the configuration file:

```
[HTMLParaStyles]
Paraname=Raw
```

where *Paraname* is the name of the placeholder paragraph format.

- (Optional) To eliminate blue borders from anchored frames, include the following setting in the configuration file:

```
[GraphAttr]
*=border="0"
```

23.10.2 Providing alternate text for a hotspot in an image map

You specify alternate text for an image-map hotspot via an attribute of the hotspot `<area>` tag. The alternate text relates to the hotspot link destination. Unlike alternate text for the `` tag, you cannot specify alternate text for the `<area>` tag with a marker or a paragraph format. And you might want to use the `title` attribute of the `<area>` tag instead of the `alt` attribute.

In this section:

§23.10.2.1 [Assigning alternate text to an image-map hotspot](#) on page 723

§23.10.2.2 [Using the title attribute for alternate text for a hotspot](#) on page 724

23.10.2.1 Assigning alternate text to an image-map hotspot

To provide alternate text for a hotspot in an image map, assign the text to the destination of the hotspot link. For example:

```
[GraphALT]
; destination or GraphicID#dest or URL dest = desired alt text
; a URL destination is the last part of the URL without extension
; ch01f853#RFstage = Tuner first stage
; IFstage = Intermediate Frequency stage
```

The text you assign becomes the content of the `alt` attribute of the hotspot `<area>` tag, unless you tell **Mif2Go** to use the `title` attribute instead; see §23.10.2.2 [Using the title attribute for alternate text for a hotspot](#) on page 724.

The destination ID must be one of the following, depending on the type of link:

<u>Type of link</u>	<u>Form of destination ID</u>
Link via message URL:	Base name of the destination HTML file, if the hotspot link is a message URL hypertext marker
Link via gotolink:	newlink marker content, if the hotspot link is a gotolink hypertext marker
Multiple links:	GraphicID followed by # followed by destination ID (one of the other two), to distinguish among multiple links to the same destination

Link via *message URL*

Suppose you use a hypertext **message URL** marker for an image-map link, with the following marker content:

```
message URL http://www.chezmoi.com/mylife.htm
```

You would assign alternate text for the hotspot to destination identifier `mylife`, which is the target file name (without path or extension):

```
[GraphALT
mylife = alternate text for hotspot
```

However, if the hypertext **message URL** marker content looks like this:

```
message URL http://www.chezmoi.com/mylife.htm#siblings
```

You would assign alternate text as follows:

```
[GraphALT
mylife#siblings = alternate text for hotspot
```

Link via *gotolink*

Suppose your image-map link is a **gotolink** hypertext marker, with a destination in the same document (though not necessarily the same FrameMaker file); for example:

```
gotolink awards.fm:firstplace
```

You would assign alternate text for the hotspot to destination identifier `firstplace`, which is the target **newlink**marker content (without the FrameMaker file name):

```
[GraphALT
firstplace = alternate text for hotspot
```

Multiple links

If you have several graphics with image-map links that all point to the same destination, and you want different alternate text for one of them, prefix the destination identifier with the file name of the graphic (no extension) and a `#`. For example:

```
[GraphALT
ab34e651#firstplace = different alternate text for hotspot
```

23.10.2.2 Using the title attribute for alternate text for a hotspot

When you provide alternate text for a hotspot in an image map, by default **Mif2Go** assigns the text to the `alt` attribute of the hotspot `<area>` tag. Some browsers, notably Internet Explorer, show the text in a tooltip when you mouse over the hotspot, whether the text is assigned to the `alt` attribute or to the `title` attribute of the hotspot `<area>` tag. Other browsers, notably Firefox, show the tooltip on mouse-over only if the text is assigned to the `title` attribute of the hotspot `<area>` tag.

To have **Mif2Go** use the `title` attribute of the hotspot `<area>` tag instead of the `alt` attribute for alternate text:

```
[Graphics]
; UseTitleForAlt = No (default) or Yes (use title attribute
; instead of alt for alternate text
UseTitleForAlt=Yes
```

When `UseTitleForAlt=Yes`, **Mif2Go** assigns any `alt` text you specify for hotspots in image maps in the `[GraphALT]` section to the `title` attribute of the hotspot `<area>` tag, and includes an empty `alt` attribute for W3C validation.

Note: If you use the `[GraphALT]` section to assign alternate text to `` tags (see §23.8 [Providing \(or omitting\) alternate text for images](#) on page 718), that text gets transferred to the `` tag `title` attribute, and you lose the content of the `` tag `alt` attribute.

`UseTitleForAlt` affects only `alt` content added for `` tags and hotspot `<area>` tags in the `[GraphALT]` section, not `alt` content added for `` tags via markers or via the `[HTMLParaStyles]` `Alt` property.

23.10.3 Specifying jumps from image maps in framesets

If you are using the image map in a frameset, you can target jumps from the image map to the frames you want in either of the following ways:

- Associate the format in effect at the image map's anchor (not the formats in the individual hotspot text frames) with a particular frame in the [Targets] section (see §13.14 [Using framesets](#) on page 450).
- Associate the file to which the jumps are going with a frame name in the [TargetFiles] section.

You can specify a default target for all jumps not otherwise associated with a frame:

```
[HTMLOptions]
; DefaultTarget = name of target to use
; for all jumps not otherwise set
;DefaultTarget=_top
```

23.11 Supplying a background image or watermark

To provide a background image or a watermark, you can assign values to <body> attributes in the configuration file; for example:

```
[Attributes]
body= bgcolor="white" background="yourimage.jpg"
```

If you are targeting only Internet Explorer (as for HTML Help), to keep the image from scrolling with the text you could add:

```
... bgproperties="fixed"
```

All attributes and values must be on the same line, regardless of line length.

A better alternative would be to use CSS. There you could also specify that the image is to be centered, not tiled, which is probably what you would want for a watermark:

```
body { background-color: white ;
        background-image: url(yourimage.jpg) ;
        background-repeat: no-repeat ;
        background-attachment: fixed ;
        background-position: center
      }
```

or just:

```
body { background: white url(yourimage.jpg) no-repeat center fixed }
```

23.12 Converting equations for HTML

Mif2Go uses the FrameMaker graphics export filters to convert equations, even if your project does not use those filters to convert other graphics for HTML. **Mif2Go** produces equation files named the same way as other graphics files; see §5.7.4.1 [Naming files produced by FrameMaker export filters](#) on page 133.

The same export format is used for all equations; the default format for HTML output is JPEG:

```
[Setup]
; GraphicExportFormat = BMP,TIFF,WMF,JPEG,PNG,EPS,PICT,CGM,GIF,IGES
GraphicExportFormat=JPEG
```

Mif2Go scales equations up 25%, making them easy to read, but not so big as to interfere with the layout:

```
[Setup]
; EquationExportDPI = number (from 50 to 1200, default 120)
EquationExportDPI=120
; EquationFrameExpand = percentage of original size (default 125)
EquationFrameExpand=125
```

To specify the file extension to use for exported graphic equation files:

```
[Options]
; EqSuffix = suffix used by Frame for equation files (no period)
EqSuffix=bmp
```

You need to specify an extension *only* if you also set the following option:

```
[Setup]
UseGraphicFileID=Yes
```

See §5.9 [Converting equations](#) on page 136 for more information.

24 Converting tables to HTML

HTML tables are rendered quite differently from FrameMaker tables. **Mif2Go** correctly renders table cells that span rows or columns, and skips rows that are conditioned out.

Topics include:

- §24.1 [Assigning properties to tables](#) on page 727
- §24.2 [Defining sets of tables](#) on page 728
- §24.3 [Specifying table structure](#) on page 730
- §24.4 [Specifying table attributes](#) on page 735
- §24.5 [Positioning tables, table titles, and table footnotes](#) on page 746
- §24.6 [Using macros to control table properties](#) on page 748
- §24.7 [Converting tables to paragraphs](#) on page 753

Mif2Go supports WAI (Web Accessibility Initiative) markup for HTML tables; see:

- §25 [Generating WAI markup for HTML](#) on page 755
- §26 [Identifying HTML table structure for WAI](#) on page 763
- §27 [Marking HTML table cells for WAI](#) on page 775

24.1 Assigning properties to tables

Start by specifying default values for properties of all tables in your document; then, if necessary, override these default values for selected tables. You can set most default table properties in the [Tables] and [Attributes] sections of the configuration file, though a few settings for tables have no document-wide defaults.

Note: Attribute settings for tables, table rows, and table cells may be browser dependent; those settings override any values generated by **Mif2Go**.

In this section:

- §24.1.1 [Understanding which table features can be converted](#) on page 727
- §24.1.2 [Understanding precedence of assignment methods](#) on page 728
- §24.1.3 [Overriding default table and cell properties and attributes](#) on page 728

24.1.1 Understanding which table features can be converted

Not every FrameMaker table feature has a corresponding HTML attribute. If the tables in your FrameMaker document use a lot of custom ruling and shading, **Mif2Go** might not be able to translate some of those properties to HTML, because HTML lacks attributes needed to implement them. For example, ruling properties can be converted only at the <table> level, using border, frame, and rules; and the last two attributes are rendered only by Internet Explorer. Even if you target only Internet Explorer, choices are limited.

Do it with CSS

You can represent some custom table properties with CSS. The best way is to apply class attributes to table cells. For a paragraph format used consistently within those cells, you can specify a CellAttribute property for the format in [HTMLParaStyles], and specify the class name in [StyleCellAttribute]; otherwise, use **CellClass** markers that specify the class name. See §24.4.6 [Specifying attributes for table cells](#) on page 738.

You must hand-edit the CSS to specify ruling and background-color settings. Or, if you have set [CSS]WriteCssStylesheet=Always, you can include the settings in

configuration section [CSSStartMacro] or [CSSEndMacro]; see §22.8.4 [Overriding styles in Mif2Go-generated CSS files](#) on page 700.

Be sure to verify that the settings you have in mind work as intended with all browsers you find important. Not all browsers support CSS 2 the same way.

24.1.2 Understanding precedence of assignment methods

Many settings for selected tables (and for table rows and cells) can be specified several ways. [Table 24-1](#) lists the assignment methods, in order of precedence. When you specify a value for the same property in more than one way for a given table or cell, the value specified by the method with the highest precedence is the value that takes effect in HTML output. In some cases, multiple assignments of the same value result in duplicate HTML code. When this happens, the assignment with the higher precedence takes effect, because that assignment appears first in the output.

Table 24-1 Precedence of table and cell property assignment methods

Precedence	Type	Assignment method	Ref.
Highest	FrameMaker custom marker	Content of Config or HTMConfig marker, or marker whose name starts with Table , Row , or Cell and ends with an attribute name	24.4.4
	Mif2Go macro	Code in [Table*Attributes] or [Table*Macros]	24.6
	FrameMaker table or cell property	Properties assigned via Table Designer, or via Custom Ruling and Shading (<i>where corresponding HTML attributes exist</i>)	Frame-Maker Help
	Configuration setting	Settings in [TableAccess] or [TableAttributes]	24.3.2.6 , 24.4.1
	Configuration setting	Settings in [Attributes]	24.4.1
Lowest	Configuration setting	Settings in [Tables]	24.3.2 , 24.4.8

24.1.3 Overriding default table and cell properties and attributes

To override table structure properties, use settings in the [TableAccess] section; see §24.3.2.6 [Overriding row and column group settings](#) on page 733.

To override table display attributes, use settings in the [TableAttributes] section; see §24.4.2 [Overriding attributes for selected tables](#) on page 736.

To fine-tune properties for selected tables or cells, use markers or macros; see §24.4.4 [Using markers to assign attributes to tables, rows, or cells](#) on page 737 and §24.6 [Using macros to control table properties](#) on page 748.

24.2 Defining sets of tables

You can set table-specific properties, and specify overrides to table defaults, according to TableID (not recommended), table format, or table group; or you can use wildcards to make any table-specific setting apply to all or a subset of tables:

TableID: Mif2Go FileID combined with FrameMaker ObjectID for the table. See §24.2.1 [Determining the TableID](#) on page 729.

- Table format:* FrameMaker format name used for the table; make sure all your table formats are in the FrameMaker table catalog. See §24.2.2 [Creating table groups](#) on page 729.
- Table group:* A group of tables you define, using configuration markers or TableIDs and table formats to specify group membership. See §24.2.2 [Creating table groups](#) on page 729.
- Wildcard set:* An informal set of tables identified with wildcards. See §24.2.3 [Using wildcards to specify table sets](#) on page 730.

24.2.1 Determining the TableID

Use this method only for short-lived documents. FrameMaker TableIDs are not necessarily preserved when a document is moved to a different version of FrameMaker, and possibly not even when a new template is applied.

To determine the TableID for a particular table, in FrameMaker do the following:

1. Find the table's FrameMaker TableID:
 - 1.1. Click in the table heading.
 - 1.2. Without moving the mouse, Shift-click.
 - 1.3. Look at the FrameMaker status bar; it should show an entry of the form `TableID=nnnnnnn`.
2. Find the FileID for the file containing the table:
 - 2.1. Look in file `mif2go.ini`, which is in the same directory as your document.
 - 2.2. Find under `[FileIDs]` the entry for your FrameMaker file; these entries are of the form: `fmfile=aa`.
3. Combine **Mif2Go** FileID and FrameMaker TableID to get the **Mif2Go** TableID: `aanNNNNNN`.

For example, if the **Mif2Go** FileID is `bb` and the FrameMaker TableID is `123412`, the **Mif2Go** TableID would be `bb123412`. See §5.3.1 [Understanding how Mif2Go creates identifiers](#) on page 117.

24.2.2 Creating table groups

You can assign group names to tables, and then apply properties to all tables in the group with a single setting. Each table can belong to one table group. You can create table groups two ways:

[Create table groups in the configuration file](#)

[Create table groups with configuration markers.](#)

Create table groups in the configuration file

To create table groups and assign tables to groups in the configuration file:

```
[TableGroup]
; TableID or format = group name used in other Table sections for ID
; the filter first looks for TableID, then group, then table format
TableID = tablegroupname
FM table format = tablegroupname
```

For example, suppose your document contains table types you use for charts (*FormatA* and *FormatB*), and others you use as containers for text frames (*FormatC* and *Unruled*). Suppose you used one *Format C* table for a chart instead of a text frame. You could group the renegade table with the other chart-type tables by specifying its TableID:

```
[TableGroup]
FormatA=charts
```



```
FormatB=charts
aa654321=charts
FormatC=textframes
Unruled=textframes
```

Create table groups with configuration markers

Another way to create a table group, or assign tables to existing table groups, is to insert configuration markers in the tables in your FrameMaker document, with content as follows:

```
[TableGroup]=tablegroupname
```

This way you can avoid having to look up TableIDs and FileIDs. See §33.2.9.3 [Overriding table properties for HTML](#) on page 928.

24.2.3 Using wildcards to specify table sets

You can specify an informal group of tables by using wildcards with partial TableID, table format, or table-group names; see §4.6 [Using wildcards in configuration settings](#) on page 106. **Mif2Go** uses the first entry in a section that matches for each table, so put the exceptions before the general case. For example:

```
[TableAfterMacros]
af123456=
ac*=<br><br>
*=<br>
```

These settings would result in the following:

- no spacing after table af123456
- double spacing after all tables with FileID ac
- single spacing after all other tables.

24.3 Specifying table structure

If the tables in your document are complex, or if you have not used FrameMaker-defined Heading and Footing rows for your tables, you might need to specify which cells belong to headers or footers. Also, you might want to specify whether **Mif2Go** should generate any or all of the following HTML tags for your tables: <colgroup>, <th>, <thead>, <tfoot>, and <tbody>.

The settings described in this section apply to all tables in your document. You can override them for selected tables with [TableAccess] settings; see §24.3.2.6 [Overriding row and column group settings](#) on page 733 and §24.3.3.2 [Specifying different header and footer counts for selected tables](#) on page 735.

In this section:

- §24.3.1 [Choosing the table structure model](#) on page 730
- §24.3.2 [Identifying row and column groups and header cells](#) on page 731
- §24.3.3 [Identifying table headers and footers](#) on page 734

24.3.1 Choosing the table structure model

By default, **Mif2Go** converts tables to HTML using the HTML table model, and converts tables to XML using the CALS table model. To specify the CALS table model for HTML output:

```
[Tables]
; UseCALSModel = No (HTML default) or Yes (XML default)
UseCALSModel = Yes
```

When `UseCALSTableModel=Yes`, **Mif2Go** uses the CALS table model to convert tables. This is the default for generic XML, DocBook XML, and DITA XML.

When `UseCALSTableModel=No`, **Mif2Go** uses the HTML table model to convert tables. This is the default for HTML and XHTML.

24.3.2 Identifying row and column groups and header cells

In this section:

§24.3.2.1 [Using browser-dependent HTML tags for tables](#) on page 731

§24.3.2.2 [Designating table header cells](#) on page 731

§24.3.2.3 [Enumerating table column groups](#) on page 732

§24.3.2.4 [Wrapping table row groups](#) on page 732

§24.3.2.5 [Positioning table footer rows \(deprecated\)](#) on page 733

§24.3.2.6 [Overriding row and column group settings](#) on page 733

24.3.2.1 Using browser-dependent HTML tags for tables

Some browsers might not support some HTML tags for tables, such as `<colgroup>`, `<th>`, `<thead>`, `<tfoot>`, and `<tbody>`. By default, **Mif2Go** does not use these tags when converting tables, because browsers that do not support them might crash, or might not show the tables.

You can specify settings in the `[Tables]` section to direct **Mif2Go** to use these HTML table tags. [Table 24-2](#) shows the settings available. These settings apply to all tables in your document. To override a setting for one or more tables, see §24.3.2.6 [Overriding row and column group settings](#) on page 733.

Table 24-2 Browser-dependent HTML tags for tables

[Tables] setting	Default value	Purpose
<code>UseTbHeaderCode</code>	No	Use <code><th></code> for header cells; default is to use <code><td></code> for all cells
<code>ColGroupElements</code>	No	List <code><colgroup></code> elements before first table row; enables <code>scope="colgroup"</code> , each <code>ColGroup</code> head starts a new <code><colgroup></code>
<code>HeadFootBodyTags</code>	No	Wrap table rows in <code><thead></code> , <code><tfoot></code> , and <code><tbody></code> groups; enables <code>scope="rowgroup"</code> , each <code>RowGroup</code> head starts a new <code><tbody></code>

24.3.2.2 Designating table header cells

The default for HTML tables generated by **Mif2Go** is not to use the `<th>` tag to distinguish header cells from body cells. However, you can direct **Mif2Go** to identify header cells:

```
[Tables]
; UseTbHeaderCode = No (default, always use <td...>)
; or Yes (use <th...>)
UseTbHeaderCode=No
```

If you specify `UseTbHeaderCode=Yes`, **Mif2Go** generates `<th>` elements for all header cells in your tables.

24.3.2.3 Enumerating table column groups

To group table columns, table rows must be preceded by `<colgroup>` elements that determine the extent of each group:

```
[Tables]
; ColGroupElements = No (default) or Yes (to put out <colgroup>
; elements before first table row; needed to enable scope="colgroup")
ColGroupElements=No
```

This setting is intended primarily to support WAI interpretation using the WAI `scope` attribute; see §26 [Identifying HTML table structure for WAI](#) on page 763 for more information. However, you can use this setting also to add CSS `class` attributes.

Mif2Go generates `<colgroup>` elements, but not `<col>` elements. The main use of `<col>` is to give a column a `class` attribute, so you can apply column-specific formatting (borders, shading) in CSS (see §22 [Setting up CSS for HTML](#) on page 681). To use `<col>` elements, specify them in `[TableStartMacros]` (see §24.6.1 [Invoking macros around tables](#) on page 748), and supply the needed attributes there. For example:

```
[TableStartMacros]
sometable=
<colgroup>
  <col span="2" class="LeftSide" />
</colgroup>
<colgroup>
  <col class="UnitPrice" />
  <col class="MinQty" />
</colgroup>
```

If you provide your own `<colgroup>` and `<col>` elements this way, either set `ColGroupElements=No` (for all tables), or override `ColGroupElements` for those tables where you supply these elements; see §24.3.2.6 [Overriding row and column group settings](#) on page 733.

24.3.2.4 Wrapping table row groups

To group table rows, the rows must be wrapped in elements that distinguish header, footer, and body rows, and that provide a way to group body rows. By default, **Mif2Go** wraps table rows in groups.

To prevent **Mif2Go** from wrapping table row groups:

```
[Tables]
; HeadFootBodyTags = Yes (default, wrap table rows in <thead>,
; <tfoot>, and <tbody> groups, to enable scope="rowgroup") or No
HeadFootBodyTags = No
```

*Create header,
footer, and body
sections*

When `HeadFootBodyTags=Yes`, **Mif2Go** wraps table rows with `<thead>`, `<tbody>`, and `<tfoot>` tags, as follows:

- All rows that are FrameMaker-defined Heading rows, or that are included in the table header by row count, are wrapped in `<thead>...</thead>`.
- All rows that are FrameMaker-defined Footing rows, or that are included in the table footer by row count, are wrapped in `<tfoot>...</tfoot>`.
- All remaining rows are wrapped in `<tbody>...</tbody>`.

This is intended primarily to support WAI interpretation using the WAI `scope` attribute; see §26 [Identifying HTML table structure for WAI](#) on page 763 for more information. However, you can also use this setting also to add CSS `class` attributes.

24.3.2.5 Positioning table footer rows (*deprecated*)

W3C specifies that a `<tfoot>` element, if present, must immediately follow the `<thead>` element, before any `<tbody>` elements; for more information, see:

<http://www.w3.org/TR/1999/REC-html401-19991224/struct/tables.html#h-11.2.3>

By default, that is where **Mif2Go** puts `<tfoot>` elements in your HTML output. Any compliant HTML 4.x or XHTML 1.x browser should support this positioning; require it, even, and fail to display the table otherwise.

When `HeadFootBodyTags=Yes` (the default; see §24.3.2.4 [Wrapping table row groups](#) on page 732), but you want to guarantee that, for some legacy viewers, table footers will appear in HTML output at the bottom of your tables (even though this apparently flies in the face of the W3C specification), you can also specify `FootTagLast=Yes`:

```
[Tables]
; FootTagLast = No (default, put after thead) or Yes
FootTagLast=Yes
```

However, this setting is deprecated, and should not be needed for current browsers and HTML viewers.

24.3.2.6 Overriding row and column group settings

You can override the default value of `HeadFootBodyTags` or `ColGroupElements` for table groups, for tables of a certain `FrameMaker` format, and for individual tables. You can even use wildcards to specify tables that are not explicitly grouped:

```
[TableAccess]
; table ID = method list (overrides default in [Tables])
; Can override HeadFootBodyTags with HFBTags, ColGroupElements with
; CGElems.
```

You can prefix either setting with `No` to turn that setting off for selected tables. For example:

```
[TableAccess]
aa123456=NoHFBTags
ac254360=HFBTags
Group5=HFBTags NoCGElems
FormatA=CGElems
```

You can turn these settings off if you are creating your own groups (especially for `ColGroupElements`, essential if you want to add `class` attributes). If you turn off a setting that is required by other settings, the presumption is that you are supplying the attributes yourself another way, such as via macros or JavaScript.

For example, suppose you have specified `ColGroupElements=Yes`, but you want to “roll your own” column groups for table `aa123456`, and include CSS `class` attributes:

```
[TableStartMacros]
; table ID = text of macro to put after <table> tag before first <tr>
; This is where a set of custom <colgroup> elements would go.
aa123456=
<colgroup>
  <col class="FirstCol"></col>
  <col class="SecondCol"></col>
</colgroup>
<colgroup span="5" class="DataSet">
  <col class="DataFirstCol"></col>
  <col class="DataSecondCol"></col>
  <!-- three missing col tags get class DataSet -->
</colgroup>
```

You would also specify the following:

```
[TableAccess]
aa123456=NoCGElems
```

24.3.3 Identifying table headers and footers

In this section:

§24.3.3.1 [Specifying default header and footer counts for all tables](#) on page 734

§24.3.3.2 [Specifying different header and footer counts for selected tables](#) on page 735

24.3.3.1 Specifying default header and footer counts for all tables

You can specify how many column-header rows, row-header columns, and footer rows the tables in your document typically have. These settings are intended primarily to support WAI interpretation; see §26 [Identifying HTML table structure for WAI](#) on page 763 for more information. [Table 24-3](#) shows the settings.

Table 24-3 Default counts of table header rows/columns and footer rows

[Tables] setting	Default value	[TableAccess] override	Purpose
TableHeaderCols	0	HColsN	Number of columns (counting from the left) to use for row headers
TableHeaderRows	0	HRowsN	Number of rows (counting from the top) to use for column headers, including rows designated Header in FrameMaker
TableFooterRows	0	FRowsN	Number of rows (counting from the bottom) to treat as footer rows for row-grouping purposes, including rows designated Footer in FrameMaker; significant only when HeadFootBodyTags=Yes

Use these settings to establish document-wide defaults for the number of columns in row headers, the number of rows in column headers, and the number of rows in table footers.

```
[Tables]
; TableHeaderCols = count of cols in which to make td -> th,
; counting from left at the start of each row in the table
TableHeaderCols=0
; TableHeaderRows = count of rows in which to make td -> th,
; counting from the top of the table
TableHeaderRows=0
; TableFooterRows = count of footer rows from bottom of the table,
; significant only when RowGroupIDs = Yes.
TableFooterRows=0
```

For example, if your tables typically have two FrameMaker-defined header rows, and you want the first body row in most of them to be considered a header row also, you would set `TableHeaderRows=3`. To designate cells in the first column as row headers, set `TableHeaderCols=1`.

Note: You need the `TableHeaderRows` and `TableFooterRows` settings only if some header/footer rows are misclassified as body rows in FrameMaker. If you consistently use FrameMaker-defined *Header* and *Footer* rows for the headers and footers in your tables, you do not need either of these settings. For DITA output, see §15.6.2 [Marking table footer rows for future reference](#) on page 511.

24.3.3.2 Specifying different header and footer counts for selected tables

You can override the default number of header columns, header rows, or footer rows with `[TableAccess]` settings; for example:

```
[TableAccess]
; table ID = method list (overrides default in [Tables]); can
; include HColsN and HRowsN, where N is the number of cols or rows
; to make td -> th (overrides TableHeaderCols and TableHeaderRows),
; and FRowsN to override TableFooterRows.
aa132446=HCols1
aa133564=HCols2 HRows3 FRows2
FormatA=FRows1
```

These `[TableAccess]` settings have the following effects:

<code>HColsN</code>	Treats cells in the first <i>N</i> columns (counting from the left) as row headers; tags the cells <code><th></code> if <code>[Tables]UseTbHeaderCode=Yes</code> . See §24.3.2.2 Designating table header cells on page 731).
<code>HRowsN</code>	Treats cells in the first <i>N</i> rows (counting from the top) as column headers; tags the cells <code><th></code> if <code>[Tables]UseTbHeaderCode=Yes</code> . See §24.3.2.3 Enumerating table column groups on page 732).
<code>FRowsN</code>	Treats the last <i>N</i> rows (counting from the bottom) as footer rows; wraps them in a <code><tfoot>...</tfoot></code> element if <code>[Tables]HeadFootBodyTags=Yes</code> . See §24.3.2.4 Wrapping table row groups on page 732).

You could use these settings to specify the structure of every table in your document. However, if all or most of the tables in your document happen to need `HCols1` (for example), it is easier to specify `[Tables]TableHeaderCols=1`, and use the `[TableAccess]` settings only for exceptions.

If `[Tables]UseTbHeaderCode=No` (the default setting), even if you specify `HColsN` or `HRowsN`, the affected cells are tagged `<td>` instead of `<th>`; however, all **Mif2Go** settings for header cells work just as though the cells were tagged `<th>`.

24.4 Specifying table attributes

When you set up an HTML or XML conversion project in FrameMaker, **Mif2Go** includes default values in the configuration file for certain attributes of HTML table tags, based on set-up options (see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425) and on properties of the tables in your FrameMaker document. You can change some of these values at set-up time via the *Set Up HTML/XML Project* dialog.

You can also exclude auto-generated attributes from HTML or XML output, and have **Mif2Go** include only those for which you specify explicit values.

In this section:

- §24.4.1 [Specifying attributes for all tables](#) on page 736
- §24.4.2 [Overriding attributes for selected tables](#) on page 736
- §24.4.3 [Assigning a CSS class to a table](#) on page 737
- §24.4.4 [Using markers to assign attributes to tables, rows, or cells](#) on page 737
- §24.4.5 [Specifying attributes for table rows](#) on page 737
- §24.4.6 [Specifying attributes for table cells](#) on page 738
- §24.4.7 [Eliminating automatically generated attributes](#) on page 739

§24.4.8 [Adjusting borders, cell spacing, and cell padding](#) on page 739

§24.4.9 [Determining the width of table columns](#) on page 741

§24.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 744

§24.4.11 [Using shading and color in tables](#) on page 745

See also:

§24.1.2 [Understanding precedence of assignment methods](#) on page 728

§24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750

24.4.1 Specifying attributes for all tables

You can specify default HTML attributes for most table-related tags, although values for some attributes might not be recognized by some browsers.

To specify default attributes for all tables:

```
[Attributes]
; HTML element = attributes (macro) to set
```

You can specify attributes here for the following tags: `body`, `table`, `tr`, `td`, `th`, `thead`, `tfoot`, and `tbody`. For example:

```
[Attributes]
table= rules="rows"
th= align="left" bgcolor="yellow"
td= valign="top"
```

List all attributes for a given tag on one line, even if that line is very long. Also see §24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750.

*No border,
cellspacing, or
cellpadding*

If you list attributes for the `<table>` tag, do *not* include `border`, `cellspacing`, or `cellpadding`; if you do, **Mif2Go** writes duplicate assignments for any of these you specify in `[Attributes]`. Use one of the following instead:

- Settings in the `[Tables]` section; see §24.4.8 [Adjusting borders, cell spacing, and cell padding](#) on page 739.
- Attributes in the `[TableAttributes]` section; see §24.4.2 [Overriding attributes for selected tables](#) on page 736.

Also see §24.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 740.

24.4.2 Overriding attributes for selected tables

To specify HTML `<table>` attributes for a single table or a group of tables:

```
[TableAttributes]
; Table ID = text (macro) to put inside table element, overrides
; settings in [Tables] for Border, Spacing, and Padding, and
; [Attributes] for table
SomeTable = attribute="value"
```

On the left of the `=` sign you can specify a TableID, a table format name, or a table group name, and you can use wildcards in the name. See §24.2.2 [Creating table groups](#) on page 729.

On the right of the `=` sign you can include any arbitrary HTML, even macros (see §28 [Working with macros](#) on page 787) and JavaScript (perhaps something like `onmouseover="javascript:dosomething(now)"`). Just keep it all on the same line.

For example, to maintain table cell borders more or less as they were in FrameMaker, rather than use a global setting for all tables (as you would in the [Tables] section or in the [Attributes] section), you can set the borders based on the table format name:

```
[TableAttributes]
FormatA= border="0" cellspacing="2" cellpadding="1"
```

Values you specify in the [TableAttributes] section override corresponding settings in the [Tables] section, and also override any attributes you assign to the table element in the [Attributes] section. However, see §24.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 740 for special constraints on specifying values for border, cellspacing, and cellpadding.

See also:

§24.1.2 [Understanding precedence of assignment methods](#) on page 728

24.4.3 Assigning a CSS class to a table

The default CSS class for a table is the FrameMaker table format name. To assign a different class:

```
[TableClasses]
; Table format name = class to use (default is based on name)
; For XML, the class is used as the tag name by default.
TableFormatName = classname
```

You could also use the [TableAttributes] section to assign a different class name to one or more tables. However, that method is deprecated.

See also:

§22.7.4 [Assigning CSS classes to table formats](#) on page 694

24.4.4 Using markers to assign attributes to tables, rows, or cells

If you give a FrameMaker custom marker type a name that starts with **Table**, **Row**, or **Cell**, **Mif2Go** uses the content of the marker for the value of whatever HTML attribute is designated by the rest of the marker-type name, and puts the attribute and its value in the generated <table>, <th>, <tr>, or <td> tag. An attribute value assigned with a custom marker takes precedence over values of the same attribute assigned any other way; see [Table 24-1](#) on page 728.

For example, to guarantee that a certain table cell is top-aligned in HTML regardless of its properties in FrameMaker or any properties assigned in the configuration file, you could insert a marker of type **CellValign** in the cell in FrameMaker, and make the content of that marker `top`. In HTML, the resulting tag for that cell would be `<td valign="top">`.

See also:

§25.1.3 [Creating custom markers for WAI attributes](#) on page 756

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

24.4.5 Specifying attributes for table rows

You can specify attributes for the <tr> element with any of the following:

[Paragraph format](#)

[Table format](#)

[Attribute marker.](#)

See also:

§24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750.

Paragraph format To specify row attributes based on the paragraph format of the content of cells in the row, you can use settings such as the following:

```
[HTMLParaStyles]
; RowAttribute inserts the contents of [StyleRowAttribute] into the
; start tag of the enclosing table row (ignored outside tables).
CellBody2 = RowAttribute

[StyleRowAttribute]
; doc style = attribute to insert in enclosing table row start tag,
; used in addition to other row attributes given under [Table...]
CellBody2 = bgcolor="yellow"
```

These settings would assign background color yellow to every row that contains a *CellBody2* paragraph, in every table. To apply the settings only to some tables, you can turn these settings on and off around specific tables; see §33.2.9.1 [Overriding paragraph and character format properties](#) on page 926.

Table format To base row attributes on the table format:

```
[TableRowAttributes]
FormatA = bgcolor="yellow"
```

This setting assigns background color yellow to every row in every *FormatA* table. See §24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750.

You can use a macro for the assignment in [TableRowAttributes].

Attribute marker To assign an attribute to an individual row, place a marker of type **RowAttr** inside the cell, where **Attr** is the name of the attribute. The marker content is just the attribute value, without quotes. For example, to assign a class to an individual row, place a marker of type **RowClass** inside any cell in the row. See §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.

24.4.6 Specifying attributes for table cells

To specify attributes for the <td> and <th> elements, you can use any of the following:

[Paragraph format](#)

[Table format](#)

[Attribute marker.](#)

Also see §24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750.

Paragraph format If all the cells to which you want to assign a particular attribute or set of attributes contain text in a particular paragraph format, for example *CellBody*, you can use settings such as the following:

```
[HTMLParaStyles]
; CellAttribute inserts the contents of [StyleCellAttribute] into
; the start tag of the enclosing table cell (ignored outside tables).
CellBody = CellAttribute

[StyleCellAttribute]
; doc style = attribute to insert in enclosing table cell start tag,
; used in addition to other cell attributes given under [Table...]
CellBody = class="mycellstyle"
```

See §26.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 768. You can use a macro for the assignment in [StyleCellAttribute].

Table format To base cell attributes on the table format:

```
[TableCellAttributes]
FormatA = class="mycellstyle"
```

See §24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750. You can use a macro for the assignment in [TableCellAttributes].

Attribute marker To specify attributes for an individual cell, place a marker of type **CellAttr** inside the cell, where **Attr** is the name of the attribute. The marker content is just the attribute value, without quotes. For example, to assign a CSS class to an individual row, place a marker of type **CellClass** inside the cell, and make the marker content the name of the class. See §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.

24.4.7 Eliminating automatically generated attributes

Mif2Go automatically generates the following attributes for tables in HTML (but not XML), based in part on properties of the tables in your FrameMaker document:

[border](#), [cellspacing](#), [cellpadding](#)
[align](#), [valign](#)
[bgcolor](#)

You might not want these automatically generated attributes in HTML output, especially if you are using CSS to control table appearance.

To exclude automatically generated attributes from HTML output, while preserving any of the same attributes you specify explicitly in the configuration file or in markers:

```
[Tables]
; TableAttributes = Yes (default for HTML) or No (default for XML)
TableAttributes=No
; CellAlignAttributes = Yes (default for HTML) or No (default for XML)
CellAlignAttributes=No
; CellColorAttributes = Yes (default for HTML) or No (default for XML)
CellColorAttributes=No
```

border, cellspacing, cellpadding When TableAttributes=No, automatically generated border, cellspacing, and cellpadding attributes are excluded from HTML output; see §24.4.8.3 [Excluding border, cellspacing, and cellpadding attributes](#) on page 741.

align, valign When CellAlignAttributes=No, automatically generated align and valign attributes based on FrameMaker table properties are excluded from HTML output.

bgcolor When CellColorAttributes=No, automatically generated bgcolor attributes based on FrameMaker table properties are excluded from HTML output.

Excluded from XML output If you are generating XML, by default **Mif2Go** excludes these automatically generated attributes; however, **Mif2Go** still includes any of these attributes that you specify explicitly in markers or in either of the following sections:

- [Attributes]; see §24.4.1 [Specifying attributes for all tables](#) on page 736)
- [TableAttributes]; see §24.4.2 [Overriding attributes for selected tables](#) on page 736.

See §14.2 [Setting up a generic XML project](#) on page 459 and §14.4.3 [Eliminating HTML attributes and tags for generic XML](#) on page 463.

24.4.8 Adjusting borders, cell spacing, and cell padding

Mif2Go gets a little weird around table-cell borders, spacing, and padding. Unless you change their values at set-up time via the *Set Up HTML/XML Project* dialog (see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425), or suppress their

appearance entirely (see §24.4.7 [Eliminating automatically generated attributes](#) on page 739), the automatically generated default values for table borders, cell spacing, and cell padding are as follows:

```
<table border="3" cellpadding="6" cellspacing="2">
```

These values are specified in pixels. In FrameMaker, table borders are additive to the cell settings for the enclosed paragraph format; there is no real equivalent in HTML settings.

In this section:

§24.4.8.1 [Specifying default border, cellspacing, and cellpadding values](#) on page 740

§24.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 740

§24.4.8.3 [Excluding border, cellspacing, and cellpadding attributes](#) on page 741

24.4.8.1 Specifying default border, cellspacing, and cellpadding values

To change the default border, cellspacing, and cellpadding values for all tables:

```
[Tables]
; Border, Spacing and Padding defaults for full table
Border=3
Spacing=2
Padding=6
```

The values for Border, Spacing, and Padding specified here become the values for HTML `<table>` attributes border, cellspacing, and cellpadding, respectively.

See also:

§24.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 740

§24.4.8.3 [Excluding border, cellspacing, and cellpadding attributes](#) on page 741

24.4.8.2 Taming border, cellspacing, and cellpadding settings

You can specify values for border, cellspacing, and cellpadding in any of these sections:

```
[Tables]
[Attributes]
[TableAttributes]
```

However, if you specify values in more than one section for the same attribute (that is, values that apply to the same set of tables) you might get:

[Duplicate attribute values](#)

[Missing attribute values.](#)

*Duplicate
attribute values*

If you specify table border, cellspacing, or cellpadding values in the [Attributes] section, **Mif2Go** also includes the Border, Spacing, and Padding settings in the [Tables] section, resulting in duplicate assignments, which are not valid HTML. *For these attributes, use only the [Tables] section.*

*Missing attribute
values*

If you specify border, cellspacing, or cellpadding values for a table or group of tables in [TableAttributes], *even if what you list in [TableAttributes] does not include any corresponding attributes*, for that group of tables **Mif2Go** ignores all of the following:

- the Border, Spacing, and Padding settings (if any) in [Tables]
- any border, cellspacing, or cellpadding values in [Attributes].

This means that if you set *any* of border, cellspacing, or cellpadding in the [TableAttributes] section, *you must set them all*; the entry in [TableAttributes] replaces all three. If you set border="0" and you want *any* cell padding or cell spacing,

in the same [TableAttributes] entry you must specify greater-than-zero values for cellspacing and cellpadding. If you omit an attribute, **Mif2Go** writes no value at all for that attribute, in which case the browser default would apply.

For example, if you specify:

```
[Tables]
Border=0
Spacing=3
Padding=6

[Attributes]
table= cellspacing="2"
```

For every table you would get a duplicate assignment for cellspacing:

```
<table border="0" cellspacing="3" cellpadding="6" cellspacing="2">
```

Then if you also specify:

```
[TableAttributes]
FormatA= border="2"
```

For *FormatA* tables you would get *only* a border value:

```
<table border="2">
```

24.4.8.3 Excluding border, cellspacing, and cellpadding attributes

To exclude from HTML or XML output the automatically generated border, cellspacing, and cellpadding attributes:

```
[Tables]
; TableAttributes = Yes (HTML default, to allow above values), or
; No (XML default, to exclude those while keeping any attributes
; explicitly added in the .ini or in markers).
TableAttributes=No
```

When TableAttributes=No, automatically generated border, cellspacing, and cellpadding attributes (in the [Tables] section) are excluded from HTML or XML output. However, **Mif2Go** includes any values you specify for these attributes in markers or in the following sections:

- [Attributes]; see §24.4.1 [Specifying attributes for all tables](#) on page 736
- [TableAttributes]; see §24.4.2 [Overriding attributes for selected tables](#) on page 736.

See also:

- §24.4.8.1 [Specifying default border, cellspacing, and cellpadding values](#) on page 740
- §24.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 740

24.4.9 Determining the width of table columns

By default, table columns are adaptively sized in HTML output. You can change the default sizing method at set-up time (see §13.2.2 [Choosing set-up options for an HTML or XHTML project](#) on page 425), or specify a different sizing method in the configuration file. You can also override the default sizing method for particular tables.

In this section:

- §24.4.9.1 [Specifying a method for determining table column widths](#) on page 742
- §24.4.9.2 [Overriding the default table column sizing method](#) on page 742
- §24.4.9.3 [Scaling the width of table columns via fixed sizing](#) on page 742
- §24.4.9.4 [Maintaining the width of table columns via relative sizing](#) on page 743

§24.4.9.5 [Controlling word breaks in table columns](#) on page 744

24.4.9.1 Specifying a method for determining table column widths

To specify a default sizing method for columns in all tables:

```
[Tables]
; TableSizing = Adaptive, Fixed (pixels), or Percent (of table)
TableSizing = Adaptive
```

[Adaptive table sizing](#) resizes columns individually to fit content; this is the default setting. However, if your document includes a series of tables on related subjects, you might want those tables to have a consistent look. [Relative table sizing](#) maintains the same relative column widths as in FrameMaker, and adjusts columns proportionally to fit the browser window. [Fixed table sizing](#) also maintains relative column widths, but does not adjust them to fit the browser window.

This setting affects the width attribute of table cells; it does not affect attributes of the <table> element itself. To override the default sizing method for particular tables and table groups, see §24.4.9.2 [Overriding the default table column sizing method](#) on page 742.

Adaptive table sizing

When TableSizing=Adaptive (the default), **Mif2Go** does not automatically generate any width attribute for table cells, so columns are resized to fit content. This setting is best, unless you have a compelling reason to specify Fixed or Percent.

Relative table sizing

When TableSizing=Percent, you get the same relative column widths in HTML that you have in FrameMaker. **Mif2Go** computes the width of each column as a percent of the table width in FrameMaker, and gives each table cell a width attribute with a value expressed as a percent of the full table width. See §24.4.9.4 [Maintaining the width of table columns via relative sizing](#) on page 743.

Fixed table sizing

When TableSizing=Fixed, **Mif2Go** gives each table cell a width attribute with a value expressed in pixels. This method might require users to scroll horizontally to see the whole table.

To specify a DPI value as a basis for determining the number of pixels:

```
[Tables]
; TableDPI = 96 (for Fixed, gives 624 pixels for 6.5" page)
TableDPI = 96
```

The default value of TableDPI is 96. See §24.4.9.3 [Scaling the width of table columns via fixed sizing](#) on page 742.

24.4.9.2 Overriding the default table column sizing method

To override the default table sizing method for selected tables or table groups:

```
[TableSizing]
; table ID = type of sizing: Adaptive (default), Fixed (pixels),
; Percent (of table width in FrameMaker)
TableID = Adaptive
```

This setting overrides the default method specified by [Tables]TableSizing; see §24.4.9.1 [Specifying a method for determining table column widths](#) on page 742.

24.4.9.3 Scaling the width of table columns via fixed sizing

To specify scaled widths for table columns, use the Fixed sizing method, and provide an appropriate value for TableDPI. For example:


```
[Tables]
TableSizing = Fixed
TableDPI = 96
```

This combination applies the same multiplier to each column width, for all the tables in your document. For example, if a column is 1 inch wide in FrameMaker, that column would be given width “96” in HTML; a 2-inch column would have width “192”, and so forth. The default setting, `TableDPI=96`, produces 624 pixels for a 6.5-inch-wide table, so the setting approximates the number of pixels.

To scale all columns wider or narrower, specify a correspondingly larger or smaller value of `TableDPI`. For example, `TableDPI=144` would enlarge the table by 50%.

To scale each column a different amount, pick a value of `TableDPI` that fits most of the columns, then adjust the actual column widths in FrameMaker to yield the correct column width in pixels.

To apply a different column scaling factor to selected tables, see §24.4.2 [Overriding attributes for selected tables](#) on page 736.

24.4.9.4 Maintaining the width of table columns via relative sizing

You can specify relative column widths for a table to override the widths specified in your FrameMaker document. Otherwise, use one of the settings described in §24.4.9.1 [Specifying a method for determining table column widths](#) on page 742, such as:

```
[Tables]
TableSizing = Percent
```

Suppose you have specified the following settings as defaults for all tables in your document:

```
[Tables]
TableSizing = Adaptive
Border = 1
Spacing = 0
Padding = 4
```

And suppose for one particular table format, *TwoCol*, you want relative column widths:

```
[TableSizing]
TwoCol = Percent
```

This setting would make the width of each column in each *TwoCol* table a percent of the width of that particular table; but the setting would not specify the percentage.

If what you really want is for each *TwoCol* table to have columns of equal width, instead you would specify:

```
[TableCellAttributes]
TwoCol = width="50%"
```

Naturally, this setting works only if all *TwoCol* tables have exactly two columns.

To set the width of the table itself, you could add:

```
[TableAttributes]
TwoCol = width="100%"
```

This setting would eliminate your `[Tables]` settings for border, cellpadding, and cellspacing; so you would have to add them to the attribute list for *TwoCol* tables:

```
[TableAttributes]
TwoCol = width="100%" border="1" cellpadding="4" cellspacing="0"
```

See §24.4.2 [Overriding attributes for selected tables](#) on page 736.

24.4.9.5 Controlling word breaks in table columns

Sometimes you might have a long word that FrameMaker broke to fit in a table column, but that the browser did not break, causing problems.

To force the same line breaks in HTML as in FrameMaker:

```
[Tables]
; TableWordBreak = No (default) or Yes (force line breaks in tables
;   when the break happens after a word-break hyphen)
TableWordBreak = Yes
```

This works best with Fixed columns; see §24.4.9.3 [Scaling the width of table columns via fixed sizing](#) on page 742.

24.4.10 Deciding what to do with empty paragraphs in table cells

Browsers neither shade nor apply borders to table cells that are empty, or that contain only tags but no content. By default, **Mif2Go** adds a single nonbreaking space between the opening and closing tags of each otherwise empty table-cell paragraph. This is appropriate for HTML and generic XML, but not for DITA XML or DocBook XML.

You can have **Mif2Go** do any of the following:

[Omit empty paragraph tags](#)

[Retain images in otherwise empty paragraphs](#)

[Provide content for empty paragraphs](#)

[Retain empty paragraph tags.](#)

*Omit empty
paragraph tags*

To omit empty paragraphs from table cells:

```
[Tables]
; RemoveEmptyTableParagraphs = No (default)
;   or Yes (DITA/DocBook default)
RemoveEmptyTableParagraphs = Yes
```

When RemoveEmptyTableParagraphs=Yes, paragraph tags are omitted for empty paragraphs in table cells (except for preformatted text, where tags are always preserved). If a table cell is blank in FrameMaker (contains only empty paragraphs), in HTML output that cell would consist of only <td></td>.

When RemoveEmptyTableParagraphs=No, the tags for empty paragraphs are retained in table cells.

*Retain images in
otherwise empty
paragraphs*

If a table-cell paragraph that contains no text includes an image set to *Run into Paragraph*, you can specify that the image should be retained even when you have set RemoveEmptyTableParagraphs=Yes:

```
[Graphics]
; RetainRuninImagesForEmptyParagraphs = No (default) or Yes
RetainRuninImagesForEmptyParagraphs=Yes
```

See §23.5.7 [Retaining run-in images in otherwise empty paragraphs](#) on page 713.

*Provide content
for empty
paragraphs*

To specify text content for otherwise empty paragraphs in table cells:

```
[Tables]
; EmptyTbCellContent = string to put in otherwise empty paragraphs
;   in table cells
EmptyTbCellContent = &nbsp;
```

The default value for EmptyTbCellContent is a single nonbreaking space: .

*Retain empty
paragraph tags*

To retain paragraph tags but omit text content for empty paragraphs in table cells:

```
[Tables]
RemoveEmptyTableParagraphs = No
EmptyTbCellContent =
```

When RemoveEmptyTableParagraphs=Yes, EmptyTbCellContent has no effect.

See also:

§21.3.10 [Eliminating empty paragraphs in text](#) on page 652

24.4.11 Using shading and color in tables

The row color (or column color) in a FrameMaker table is always one of the following:

- the main table background color
- if you specify colors other than As Is on the **Shading** tab in *Table Designer*, and set both **First** and **Next** counts to non-zero values:
 - the alternate-row color, if you select **Shade By Body Row(s)**
 - the alternate-column color, if you select **Shade By Column(s)**
- any cell color(s) you have specified via Custom Ruling and Shading.

Mif2Go automatically generates bgcolor attributes for HTML tables based on table colors in FrameMaker. You can exclude these automatically generated attributes, while including any bgcolor attributes you specify in the configuration file or in a marker; see §24.4.7 [Eliminating automatically generated attributes](#) on page 739.

You can have **Mif2Go** apply to HTML tables the same alternate-row or alternate-column shading you defined in FrameMaker Table Designer:

```
[Tables]
; UseAltShading = No (default)
; or Yes (alternate row/col shading as in Frame)
UseAltShading=Yes
```

Mif2Go uses the FrameMaker definition of row type to determine the extent of the table body, and not the configuration-file settings for heading/footer row counts you can specify with the [Tables] and [TableAccess] keywords described in §24.3.3 [Identifying table headers and footers](#) on page 734.

Mif2Go applies alternate-column colors at the cell level, because <colgroup> is not widely supported by browsers.

Where you have specified other colors for cells via Custom Ruling and Shading, those colors take precedence. Otherwise, if UseAltShading=No, and you defined alternate-row or alternate-column shading in a FrameMaker table, all body rows or columns of that table receive the color you specified for **First** on the **Shading** tab in *Table Designer*.

Background color

With the default setting, UseAltShading=No, if you specify the HTML bgcolor attribute for the <table> element, that background color applies to all cells. You can have **Mif2Go** apply whichever color is appropriate (which might be the alternate color if UseAltShading=Yes) in <tr> tags, as well as in cell tags:

```
[Tables]
; UseRowColor = No (default) or Yes (set bgcolor for <tr> tag)
; overridden on table ID basis by [TableUseRowColor] settings
UseRowColor=Yes
```

UseRowColor specifies whether to use <tr> color attributes. When UseRowColor=Yes, **Mif2Go** applies cell <td> color attributes only when the cell color is different from the row color; this is a size optimization. Setting UseRowColor=Yes might affect the color of the spaces between cells. If UseAltShading=No, probably that color is determined by

the page or table bgcolor value. However, **Mif2Go** ignores any value of bgcolor specified as a table attribute in the configuration file.

If you set UseRowColor=Yes, **Mif2Go** writes each table row using the color specified in your FrameMaker document (as modified by settings in [Colors]), provided that color is not “invisible”, and has a tint greater than 1%. Then, **Mif2Go** puts out a background color for each cell, if the cell color or tint is different from the row color or tint. If you specify UseRowColor=Yes it does not really make sense to specify a table background color; it will always be overridden.

If UseRowColor=No, all rows are considered to be 100% white; **Mif2Go** writes out any cell-level colors or tints that are different from white. If UseRowColor=No, and you specify a table background color in HTML via bgcolor, you are likely to have trouble unless the table background color is white (ffffff), in which case specifying it is redundant. If you have a cell where you want white, and you have a different table background color, you get the table background color in the cell instead, because in effect there is no cell background color.

The setting for UseRowColor applies to all tables in your document. To override this setting for selected tables:

```
[TableUseRowColor]
; table ID = Yes or No, overrides UseRowColor
```

See §24.2 [Defining sets of tables](#) on page 728 for ways to specify a subset of tables.

Shading **Mif2Go** follows all override settings for shading made via the FrameMaker *Custom Ruling and Shading* dialog; these settings overrule both the Table Designer settings and your UseAltShading setting.

Fill color Table fill colors are treated the same as text colors (see §13.7 [Defining and mapping colors for HTML](#) on page 438); the [Colors] mapping works the same way. **Mif2Go** specifies an HTML table background color and individual cell overrides as required, including using the correct heading/footer color for the cells in the table heading/footer rows. Choose colors and tints such that all the resulting colors are Web-safe. See §13.7.4 [Using Web-safe colors](#) on page 440 for more information.

You can use a table fill color that is just a tint. However, doing so might not give you Web-safe colors; a 100% tint of a distinct color is often a better idea. But not always; in one test case, a “highlight” color is applied (via Custom Ruling and Shading) to a set of columns, and shading is applied (via Table Designer) to alternate rows. What happens to cells that fall in both a shaded row and a highlighted column? If the alternate-row shading is a tint (for example, 50%) of the same color used for highlighting, the cells are tinted in the highlight color. But if the alternate-row shading is a different color, with 100% tint, these cells are not tinted; they are just highlighted, which looks a bit odd.

Border color The effect on border colors varies by browser; some use cell attributes, some use table attributes.

24.5 Positioning tables, table titles, and table footnotes

In this section:

§24.5.1 [Indenting tables](#) on page 747

§24.5.2 [Configuring and positioning table titles](#) on page 747

§24.5.3 [Eliminating FrameMaker table title variables](#) on page 748

§24.5.4 [Positioning table footnotes](#) on page 748

24.5.1 Indenting tables

Best practice is to use CSS to indent tables in HTML. The technique described in this section should be used only if you cannot use CSS.

Mif2Go can indent tables to match the indent used in your FrameMaker document. Because there is no indent attribute for tables in HTML, **Mif2Go** places a spacer graphic just before the table; see §23.6.3 [Indenting images](#) on page 716.

To use the spacer graphic:

```
[HTMLOptions]
; UseSpacers = No (default)
; or Yes, use to position tables and graphics
UseSpacers = Yes
```

See §23.6.3 [Indenting images](#) on page 716.

To specify the width of the spacer graphic:

```
[Tables]
; TableIndents=-1 (based on indent in FM), 0 (none), or count of
; pixels; overridden for particular tables and groups in
; [TableIndents] section
TableIndents = -1
```

When `TableIndents=-1` (the default), tables are indented the same as in your FrameMaker document.

When `TableIndents=0`, tables are not indented.

When `TableIndents=n`, where *n* is a positive integer, tables are indented *n* pixels.

To override this setting for a particular table or table group:

```
[TableIndents]
; TableID = number of pixels to indent using PixelSpacerImage
; zero prevents indent, -1 is autoindent (default action)
; overrides default set in [Tables]TableIndents for its table or group
```

For example, to indent table `af123456` by 60 pixels, and prevent any tables in file `ag` (see §5.3 [Identifying files and objects](#) on page 117) from being indented:

```
[TableIndents]
af123456=60
ag*=0
```

24.5.2 Configuring and positioning table titles

To specify use and placement of FrameMaker table titles in HTML or XML output:

```
[Tables]
; TableTitles = 0 to leave alone, 1 to put at top, 2 to put at bottom
TableTitles = 0
; UseInformaltableTag = No (default) or Yes (use when there is no
; table caption, as in DocBook)
UseInformaltableTag = No
; InternalTableCaption = Yes (default) or No (put outside table)
InternalTableCaption = Yes
; TableCaptionTag = tag for internal table captions, default "caption"
TableCaptionTag = caption
```

Some browsers do not like the `<caption>` tag inside the `<table>` tags. To satisfy those browsers, specify `InternalTableCaption=No`.

24.5.3 Eliminating FrameMaker table title variables

Generally you will want to eliminate FrameMaker table variables from HTML table titles, because HTML tables do not break the same way. If your table variables do not have the usual names, you can identify them here:

```
[Tables]
; TableContinued = No (default) to remove variable from table titles
TableContinued = No
; TableContVar = name of the variable used for table (continued)
TableContVar = Table Continuation
; TableSheet = No (default) to remove this variable from table titles
TableSheet=No
; TableSheetVar = name of the variable used for table (Sheet m of n)
TableSheetVar = Table Sheet
```

24.5.4 Positioning table footnotes

Table footnotes are handled in the same stream as text footnotes, instead of appearing at the end of the table. If you must keep table footnotes with the table, either make the table a file of its own (by splitting before and after it), or use cross references to simulate table footnotes.

To specify placement of footnotes:

```
[Tables]
; TableFootnotesWithTable = No (default) or Yes (put after separator)
TableFootnotesWithTable=No
; TableFootnoteSeparator = macro between table end tag and footnotes
;TableFootnoteSeparator=<br />
```

See also:

§21.11.1 [Configuring and placing footnotes](#) on page 671

§22.7.5 [Assigning CSS classes to text and table footnotes](#) on page 694.

24.6 Using macros to control table properties

You can fine-tune table appearance with settings that insert HTML code in precise locations in and around tables. This is a good place to use **Mif2Go** macros (see §28.9.5 [Assigning macros to graphics or tables for HTML](#) on page 827). You can even use macros to wrap a table in another table, to get special effects.

In this section

§24.6.1 [Invoking macros around tables](#) on page 748

§24.6.2 [Adding space before tables](#) on page 749

§24.6.3 [Adjusting space after tables](#) on page 749

§24.6.4 [Turning processing on and off around selected tables](#) on page 750

§24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750

§24.6.6 [Capturing table row and column counts with variables](#) on page 751

§24.6.7 [Selectively modifying table text with macros: an example](#) on page 752

24.6.1 Invoking macros around tables

You can specify macros to be invoked before, after, or in place of any table or group of tables, by assigning macros to a TableID in one of the [Table*Macros] sections:

```

[TableBeforeMacros]
; TableID = macro to put before table start, top title or indent

[TableStartMacros]
; TableID = macro to put after <table> tag before first <tr>
; This is where a set of custom <colgroup> elements would go.

[TableReplaceMacros]
; TableID = macro to use in place of table (and title and indent)

[TableEndMacros]
; TableID = macro to put just before </table>

[TableAfterMacros]
; TableID = macro to put after table end or bottom title

```

When you specify a macro or other HTML code to replace a table, any *Before*, *Start*, *End*, or *After* code or macro you assigned to that table in one of the other [Table...Macros] sections is not used.

24.6.2 Adding space before tables

To add space before all tables:

```

[TableBeforeMacros]
*=<br>

```

To add space before a specific table, use its TableID (see §24.2.1 [Determining the TableID](#) on page 729); for example:

```

[TableBeforeMacros]
ae1001207=<br>

```

To add space based on the table format, specify the format name (which is shown in the Table Designer):

```

[TableBeforeMacros]
FormatA=<br>

```

Mif2Go checks section [TableBeforeMacros] and uses the first rule that applies to the current table. This allows you to make exceptions. For example:

```

[TableBeforeMacros]
ae123456=<br><br>
InLine=
*=<br>

```

These settings would result in the following:

- double spacing before the table with TableID ae123456
- no spacing before any table with format *InLine*
- single spacing before all other tables.

See §24.6.1 [Invoking macros around tables](#) on page 748.

24.6.3 Adjusting space after tables

You can use this setting to add space after all tables:

```

[TableAfterMacros]
*=<br>

```

Or, you can use more specific wildcards or full TableIDs to adjust space in individual cases. **Mif2Go** uses the first entry in the section that matches, so put the exceptions before the general case:

```

[TableAfterMacros]
af123456=

```

```
ac*=<br><br>
*=<br>
```

These settings result in the following:

- no spacing after table af123456
- double spacing after all tables with FileID ac
- single spacing after all other tables.

See §24.6.1 [Invoking macros around tables](#) on page 748.

24.6.4 Turning processing on and off around selected tables

Suppose you use two-cell tables in FrameMaker to hold notes and warnings, with an icon in the first cell and text in the second cell. And suppose for HTML output you want to strip the table structure, discard the icon, and keep just the text from the second cell.

You can use *Before* and *After* macros for the table format to turn on and off the StripTables setting (see §24.7.2 [Removing table-specific tags from selected tables](#) on page 754) for the table format in question; in this example, *NoteTable*:

```
[TableBeforeMacros]
NoteTable = <$$[Tables]StripTables=1>

[TableAfterMacros]
NoteTable = <$$[Tables]StripTables=0>
```

Stripping a table removes only the table code; the content remains unaltered, so you still have both icon and text.

To exclude the icon(s) from HTML output, suppose you have established a graphics group for such icons (see §23.5.1.4 [Creating named groups of graphics](#) on page 710), with group name *NoteIcons*:

```
[GraphReplaceMacros]
NoteIcons = <$$nothing=1>
```

See §23.4.5 [Omitting graphics from HTML or XML output](#) on page 708.

24.6.5 Specifying row-group, row, and cell attributes with macros

You can specify HTML code to add attributes at precisely defined locations inside table row groups, rows, and cells. These attributes override the same attributes specified in the [Attributes] section:

[Row-group attributes](#)

[Row attributes](#)

[Cell attributes.](#)

*Row-group
attributes*

The following sections let you control attributes of table row groups, and allow you to use macros in a row-group element: <thead>, <tfoot>, or <tbody>.

```
[TableHeaderAttributes]
; table ID = text (macro) to put inside <thead>,
; overrides [Attributes] for <thead>

[TableFooterAttributes]
; table ID = text (macro) to put inside <tfoot>,
; overrides [Attributes] for <tfoot>

[TableBodyAttributes]
; table ID = text (macro) to put inside <tbody>,
; overrides [Attributes] for <tbody>
```

For these overrides to take effect, you must also specify either of the following:

- for all tables, any one of:


```
[Tables]
HeadFootBodyTags = Yes
AccessMethod = Scope
ScopeRowGroup = Yes
```
- for individual tables or table groups, any one of:


```
[TableAccess]
TableID = HFBTags
TableID = Scope
TableID = ScopeRowGroup
```

See also:

§24.3.2.4 [Wrapping table row groups](#) on page 732

§24.3.2.6 [Overriding row and column group settings](#) on page 733.

Row attributes The following sections let you control row attributes for all the rows in a table (or group of tables), and use macros in a row before and after other content. See also §26.1 [Identifying table rows and columns](#) on page 763 for ways to identify table rows for WAI (Web Accessibility Initiative) markup.

```
[TableRowAttributes]
; table ID = text (macro) to put inside <tr>, overrides [Attributes]

[TableRowStartMacros]
; table ID = text of macro to put on line after <tr>

[TableRowEndMacros]
; table ID = text of macro to put before </tr>
```

See also:

§24.4.5 [Specifying attributes for table rows](#) on page 737

Cell attributes The following sections let you control cell attributes for all the cells in a table (or group of tables), and use macros in a cell before and after other content.

```
[TableCellAttributes]
; table ID = text (macro) to put inside <td>, overrides [Attributes]

[TableCellStartMacros]
; table ID = text of macro to put after <td>

[TableCellEndMacros]
; table ID = text of macro to put before </td>
```

24.6.6 Capturing table row and column counts with variables

Two predefined macro variables allow you to access the numbers of columns and rows in the current table:

```
<$$_tblcols> Count of columns in the current table
<$$_tblrows> Count of rows in the current table
```

You can use these variables in macro expressions that manipulate or make use of table properties. For example, to add a rule above and below each table by placing the rule in an extra row that spans all columns:

```
[TableStartMacros]
*=<tr><td colspan="<$$_tblcols>"><hr></td></tr>

[TableEndMacros]
*=<tr><td colspan="<$$_tblcols>"><hr></td></tr>
```

See also:

§24.6.1 [Invoking macros around tables](#) on page 748

[§28.3.4 Using predefined macro variables](#) on page 800

24.6.7 Selectively modifying table text with macros: an example

Suppose the following:

- Some columns in your tables have text that you want bold in HTML, even though in FrameMaker the text is not differentiated with a character format or an override.
- The columns in question all have column headings that include the word “Fields”.
- The paragraph format for table body cells is *CellBody*, and the format for column headings is *CellHeading*.

To achieve selective bolding, you can use macros to assign different class attributes to paragraph format *CellBody*, based on the content of each column heading.

Use macro variables to identify table columns

Create two macro variables to hold column numbers; for example, `$$ColNum` and `$$FieldColNum`. `$$ColNum` counts the columns in a table, and `$$FieldColNum` holds the column number of any column whose heading contains the word “Fields”.

If you are not using table macros for any other purpose, you can use a wildcard to specify the following macros for all tables:

```
[TableStartMacros]
; Reset $$FieldColNum for each table:
*=<$$FieldColNum = 0>

[TableRowStartMacros]
; Reset $$ColNum for each table row:
*=<$$ColNum = 0>

[TableCellStartMacros]
; Increment $$ColNum for each table column:
*=<$$ColNum++>
```

See also:

[§24.2.3 Using wildcards to specify table sets](#) on page 730

[§24.6.1 Invoking macros around tables](#) on page 748

[§28.3 Using macro variables](#) on page 795

Assign coding options to table-cell formats

Turn off the HTML paragraph tag that **Mif2Go** would otherwise automatically assign to *CellBody*, and specify macro code for both *CellBody* and *CellHeading*:

```
[HTMLParaStyles]
; CellBody formatting will be replaced by macro code:
CellBody=NoPara CodeStart CodeEnd
; CellHeading will hold column-heading content to be checked,
; and also provide the code for checking:
CellHeading=CodeStore CodeAfter

[ParaStyleCodeStart]
; Assign a macro to CellBody, so the code can exceed one line:
CellBody=<$$SelectClass>

[ParaStyleCodeEnd]
; Provide a closing tag for the class attribute:
CellBody=</p>

[ParaStyleCodeAfter]
; Use the CodeStore property assigned to CellHeading to
; capture the content of the current CellHeading paragraph,
; and also assign a macro, so the code can exceed one line:
CellHeading=<$$CellHeading><$$CheckColHead>
```

See also:

§21.3.6 [Stripping paragraph properties](#) on page 650

§28.3.2 [Assigning values to macro variables](#) on page 797

§28.9.3 [Surrounding or replacing text with code or macros](#) on page 822

§28.3.7.2 [Inserting code with the CodeStore property](#) on page 804

Check for columns that need bolding

Use a conditional expression to check the content of each *CellHeading* paragraph:

```
[CheckColHead]
; Use string operator "contains" to check the content;
; if the text sought is present, flag the column:
<$_if ($$CellHeading contains "Fields")>
  <$$FieldColNum = $$ColNum>
<$_endif>
```

Select a class attribute based on the column flag

To select a class attribute for *CellBody*, compare \$\$ColNum and \$\$FieldColNum in a conditional expression:

```
[SelectClass]
<p class="<$_if ($$FieldColNum == $$ColNum)>CellBodyBold
  <$_else>CellBody
  <$_endif>">
```

See also:

§28.6.4 [Using control structures in expressions](#) on page 815

§28.6.5 [Specifying substrings in expressions](#) on page 817

24.7 Converting tables to paragraphs

To use content stored in tables as ordinary non-table content, you can direct **Mif2Go** to remove table-specific tagging from tables in your document, leaving just cell content. You might need to do this if you have a long table and you want to split it across several HTML files, but not as a table; or if you are preparing output to be displayed in a browser that does not support HTML table tags.

In this section:

§24.7.1 [Removing table-specific tags from all tables](#) on page 753

§24.7.2 [Removing table-specific tags from selected tables](#) on page 754

§24.7.3 [Removing table-specific tags from complex tables](#) on page 754

See also:

§24.6.4 [Turning processing on and off around selected tables](#) on page 750

24.7.1 Removing table-specific tags from all tables

To strip all tables of table-specific elements:

```
[Tables]
; StripTable = No (default) or Yes to remove all table tagging while
; retaining cell content.
StripTable = Yes
```

This setting applies to all tables in your document.

When StripTable=Yes, **Mif2Go** writes out the content of each table cell in row order (top to bottom) then column order (left to right), preserving paragraph and character formats.

By default, **Mif2Go** also sets the following file-splitting options:

```
[Tables]
; AllowTbSplit = No (default)
; or Yes (allow file split for head in table)
AllowTbSplit = Yes
; AllowTbTitle = No (default) or Yes (allow title from head in table)
AllowTbTitle = Yes
```

You can override these settings; see §18.2.1 [Designating split points](#) on page 586 and §18.4.2.2 [Assigning a title with a paragraph format](#) on page 595.

24.7.2 Removing table-specific tags from selected tables

To strip tags only from selected tables, you can use either of the following methods:

[Strip table tags with configuration macros](#)

[Strip table tags with Config markers.](#)

*Strip table tags
with configuration
macros*

To remove table tags by assigning table macros to a table ID:

```
[TableBeforeMacros]
TableID = <$$[Tables]StripTable=1>

[TableAfterMacros]
TableID = <$$[Tables]StripTable=0>
```

See §24.6.1 [Invoking macros around tables](#) on page 748 and §24.6.4 [Turning processing on and off around selected tables](#) on page 750.

*Strip table tags
with Config
markers*

To remove table tags from individual tables, you can surround each table with **Config** markers:

<u>Config marker content</u>	<u>Marker placement</u>
[Tables]StripTable=1	Before the table anchor
[Tables]StripTable=0	After the table

See §33.2.8 [Overriding fixed-key configuration settings](#) on page 924.

24.7.3 Removing table-specific tags from complex tables

For complex tables, if the raw result of [Table]StripTable=Yes is not satisfactory, you might need to use macros to control placement and appearance of content. You can use table macros for this purpose, even though the content is not displayed with table tags in the output; see §24.6 [Using macros to control table properties](#) on page 748. You can also use format-related macros; see §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

25 Generating WAI markup for HTML

Mif2Go supports WAI (Web Accessibility Initiative) guidelines for authoring HTML content intended to be accessible to people with disabilities. Topics include:

§25.1 [Comparing Mif2Go markup methods for WAI](#) on page 755

§25.2 [Applying WAI markup to images](#) on page 756

§25.3 [Applying WAI markup to links](#) on page 758

§25.4 [Applying WAI markup to tables](#) on page 759

See also:

§29 [Working with FrameMaker markers](#) on page 831

For more information about WAI, see:

<http://www.w3.org/TR/1999/WAI-WEBCONTENT-19990505/>

To test the effectiveness of the WAI attributes you specify using **Mif2Go**, see:

<http://www.w3.org/WAI/eval/Overview.html>

25.1 Comparing Mif2Go markup methods for WAI

In this section:

§25.1.1 [Choosing a markup method for WAI attributes](#) on page 755

§25.1.2 [Using paragraph formats for WAI attributes](#) on page 755

§25.1.3 [Creating custom markers for WAI attributes](#) on page 756

25.1.1 Choosing a markup method for WAI attributes

Mif2Go provides the following ways to specify WAI attributes for HTML:

- Assign a **Mif2Go** property that represents a WAI attribute to a FrameMaker paragraph format, and assign a value to the property, in the configuration file. This method is supported primarily for table markup.
- Assign a WAI attribute to a FrameMaker paragraph format in the configuration file, and make the paragraph content the attribute value. This method is supported for a limited number of WAI attributes.
- Use a custom FrameMaker marker named after a WAI attribute, and make the marker text the attribute value.
- In FrameMaker 7.0 and later versions, for graphics in anchored frames, use the FrameMaker *Object Attributes* dialog to specify attributes.

Macros and macro variables can be referenced in the attributes, both from markers and from text identified as attribute content.

25.1.2 Using paragraph formats for WAI attributes

To use a paragraph-format method, you must dedicate a different paragraph format to each WAI attribute or combination of attributes that you need in your document. You can use a paragraph format for either of the following:

- a single attribute, and use multiple paragraphs (each with a different format) to apply more than one attribute to the same item in your document;
- a combination of attributes, and use a single paragraph to apply the combination.

You must include in the FrameMaker paragraph catalog all the paragraph formats you use for this purpose.

To hide WAI markup, use conditions

Using a different paragraph format for material that does not actually call for a different format in printed versions can unduly complicate document maintenance. To get around this drawback, you can use the following variation:

1. Apply the WAI paragraph format to an extra paragraph you insert just before the item that requires the markup (or in the same cell, if in a table).
2. Apply a condition, so you can hide the content of that extra paragraph in printed versions of the document.
3. In the configuration file specify the `Delete` property for the paragraph format, to exclude the extra paragraph from HTML output.

Assign WAI attributes with properties

You assign **Mif2Go** properties to paragraph formats in the `[HTMLParaStyles]` section, and you assign values to the attributes represented by those properties in `[StyleCell*]` sections. WAI table-cell attributes can be assigned this way; see §26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767.

Use special paragraphs for WAI attribute values

In the `[HTMLParaStyles]` section you assign properties that represent WAI attributes (and usually also the `Delete` property) to a paragraph format; **Mif2Go** uses the content of the paragraph as the value of the attribute.

You can use an existing paragraph format for this purpose, and omit the `Delete` property, if the format conforms to all of the following:

- The paragraph format is not used for unrelated purposes elsewhere in the document.
- Each paragraph with that format already contains text suitable for the attribute value.
- Each paragraph with that format appears just before an item that needs the attribute (with no intervening items of the same type), or in a table cell that needs the attribute.

25.1.3 Creating custom markers for WAI attributes

You can create custom FrameMaker markers to insert in (or just before) items that require WAI markup.

Marker name is significant

If you give a custom marker type a name that starts with **Table**, **Cell**, **Graph**, or **Link**, **Mif2Go** automatically makes the marker text the value of whatever HTML attribute is named by the rest of the marker-type name, and puts the attribute and its value in the generated `<table>`, `<th>`, `<td>`, ``, or `<a>` tag. This method has two advantages:

- Markers do not require entries in the configuration file.
- Markers do not clutter your paragraph catalog.

Series of markers

The text of a FrameMaker marker is limited to 256 characters. **Mif2Go** gets around that restriction by concatenating all markers for the same attribute that are inserted before the next item to which they apply. You can just add more markers of the same type, and continue the content. However, if the text of the attribute is long (such as a summary for a large and complex table), you might not want to chop it up into three or four markers; in that case, use one of the other markup methods to apply the attribute, or use macros.

25.2 Applying WAI markup to images

In this section:

§25.2.1 [Following WAI guidelines for images](#) on page 757

§25.2.2 [Assigning WAI image attributes with dedicated formats](#) on page 757

§25.2.3 [Assigning WAI image attributes with custom markers](#) on page 757

§25.2.4 Assigning WAI image attributes via the Object Attributes dialog on page 758

25.2.1 Following WAI guidelines for images

The WAI guidelines for images are intended to provide a text equivalent for every non-text element. For more information, see:

<http://www.w3.org/TR/WCAG10-HTML-TECHS/#image-text-equivalent>

Mif2Go provides settings for the following attributes:

<code>alt</code>	Short text equivalent of an image.
<code>longdesc</code>	Path to an HTML file containing text that describes the image.
<code>title</code>	Image description, for user agents that do not support <code>longdesc</code> .

You can provide `alt` and `title` attributes for an image either with a paragraph format or with a custom marker.

25.2.2 Assigning WAI image attributes with dedicated formats

You can designate a paragraph format whose content will be the text alternative for the next anchored frame in a flow. For example, suppose you use paragraph format *Figname* for this purpose:

```
[HTMLParaStyles]
; Alt makes current para content into alt attribute for next img
Figname=Alt Delete
```

Somewhere just before the image you would insert a *Figname* paragraph containing the name you want displayed as an alternate for the graphic image. Probably you would make the *Figname* paragraph conditional so it would not appear in print. The `Delete` property would exclude the paragraph (as such) from HTML output; HTML source would show the text of the *Figname* paragraph as the value for the `alt` attribute of the `` element.

For example, if you were to place a *Figname* paragraph with the content “Cat in basket” just before the image, in some browsers moving the pointer over the image would display this content in a tooltip. However, in most current browsers, the tooltip shows the content of the `title` attribute rather than the `alt` attribute; and the `alt` text is displayed only when the image is not displayed or is missing.

You can use a similar strategy to provide content for the `longdesc` attribute of the `` element; for example, using paragraph format *Figdesc* for this purpose:.

```
[HTMLParaStyles]
; Longdesc makes current para content into longdesc attribute
Figdesc = Longdesc Delete
```

The `Delete` property would exclude the paragraph from normal HTML text output.

25.2.3 Assigning WAI image attributes with custom markers

You can use markers to provide text equivalents of graphic images. The attribute value in the text of a marker applies to the next anchored frame in a flow after the marker. Each marker name must start with `Graph` and end with the name of the attribute. Valid marker names are as follows

GraphAlt	Short text equivalent of image; adds <code>alt</code> attribute.
GraphLongdesc	Path to file with image description; adds <code>longdesc</code> attribute.
GraphTitle	Long text description of image; adds <code>title</code> attribute.

For example, to provide an `alt` attribute for each graphic:marker

1. Create a FrameMaker marker type named **GraphAlt**.
2. For each graphic image:
 - 2.1. Place a **GraphAlt** marker just before the image.
 - 2.2. Type the text equivalent in the *Marker* dialog.
 - 2.3. Click **New Marker**.

25.2.4 Assigning WAI image attributes via the *Object Attributes* dialog

FrameMaker 7.0 and later versions provide a way to assign attributes to anchored frames. Select an anchored frame and choose **Object Properties...** from the right-click context menu or the FrameMaker **Graphics** menu. In the *Object Properties* dialog, click **Object Attributes...** to open the *Object Attributes* dialog. **Mif2Go** includes in HTML output any image attributes you specify in this dialog. See:

§23.7 [Specifying HTML image attributes](#) on page 718.

§31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896

25.3 Applying WAI markup to links

In this section:

§25.3.1 [Following WAI guidelines for links](#) on page 758

§25.3.2 [Assigning WAI link attribute values with dedicated formats](#) on page 758

§25.3.3 [Assigning WAI link attribute values with custom markers](#) on page 759

25.3.1 Following WAI guidelines for links

The HTML `title` attribute is commonly used in links, where it “hides” the value of the `href` attribute that is otherwise shown, replacing it with a text description of the link destination. For more information about WAI guidelines for links, see:

<http://www.w3.org/TR/WCAG10-HTML-TECHS/#links>

You can provide a `title` attribute for a link either with a paragraph format or with a custom marker; and either may reference macro variable `<$$_linksrc>`.

25.3.2 Assigning WAI link attribute values with dedicated formats

You can designate a paragraph format whose content will be the text for the following link. For example, suppose you use paragraph format *Linkname* for this purpose:

```
[HTMLParaStyles]
; LinkTitle makes current para content into title attr for next link
Linkname = LinkTitle Delete
```

In your FrameMaker document, just before the link you would insert a *Linkname* paragraph containing the name you want displayed for the link destination. Probably you would make the *Linkname* paragraph conditional so it would not appear in print. The `Delete` property would exclude the paragraph (as such) from HTML output; HTML source would show the text of the *Linkname* paragraph as the value for the `title=` attribute of the `<a>` tag.

You can use a similar strategy to assign a CSS class to the next link. For example, to use paragraph format *LinkCSS* for this purpose:

```
[HTMLParaStyles]
; LinkClass makes current para content into class attribute
; used to set the link display properties in CSS.
LinkCSS = LinkClass Delete
```

The Delete property would exclude the paragraph from normal HTML text output.

25.3.3 Assigning WAI link attribute values with custom markers

You can use markers to provide text alternatives (and other attributes) for links. The attribute applies to the next link after the marker. The marker name must start with **Link** and end with the name of the attribute:

LinkClass CSS class for the next link.
LinkTitle Descriptive title for the link destination.

For example, to provide a `title` attribute for each link:

1. Create a FrameMaker marker type named **LinkTitle** (see §29.2 [Adding custom marker types](#) on page 832).
2. For each link:
 - 2.1. Place a **LinkTitle** marker just before the link.
 - 2.2. Make the content of the marker the text you want for the title.

To assign a CSS class to a link, see §19.2.2.1 [Assigning a link class with a marker](#) on page 610.

25.4 Applying WAI markup to tables

In this section:

- §25.4.1 [Following WAI guidelines for tables](#) on page 759
- §25.4.2 [Choosing a WAI markup method for tables](#) on page 760
- §25.4.3 [Providing table summary and title information](#) on page 760
- §25.4.4 [Identifying table row and column information](#) on page 762

See also:

- §26 [Identifying HTML table structure for WAI](#) on page 763
- §27 [Marking HTML table cells for WAI](#) on page 775

25.4.1 Following WAI guidelines for tables

You can assign WAI attributes to tables, and within tables to rows, columns, and header cells. **Mif2Go** supports WAI guidelines for the following kinds of table markup:

- Providing summary information (WAI guidelines 5.1 and 5.2).
- Identifying row and column information (WAI guidelines 5.5 and 5.6).

Mif2Go provides settings for the following attributes:

abbr Abbreviation for the contents of a cell.
axis Conceptual category of cell content, provided for queries.
headers References to header-cell IDs.
id ID (name) of a header cell.
scope Rows or columns covered by a header cell.

summary	Description of a table's purpose and structure.
title	Brief description of table.

Mif2Go automatically generates markup for the following (non-WAI) attributes, based on row and column straddling in your FrameMaker tables:

colspan	Number of columns spanned (straddled) by a cell.
rowspan	Number of rows spanned (straddled) by a cell.

For information about WAI table guidelines, see:

<http://www.w3.org/TR/WCAG10-HTML-TECHS/#data-tables>

For information about using WAI table attributes, see:

<http://www.w3.org/TR/1999/REC-html401-19991224/struct/tables.html#h-11.4>

25.4.2 Choosing a WAI markup method for tables

For WAI markup that affects the table as whole, probably it does not matter which markup method you choose (see §25.1 [Comparing Mif2Go markup methods for WAI](#) on page 755). However, for markup that affects individual rows, columns, or cells, the “best” (most practical) method depends on the following characteristics of the tables in your FrameMaker document:

- **Number** (are you converting a single table, 10 tables, or 1,000 tables?)
- **Size** (are most tables on the order of two rows by three columns, or 2,000 rows by 15 columns?)
- **Diversity** (do most tables have the same structure, or do they vary widely?)
- **Complexity** (do some tables have more than two dimensions of data, or multiple row or column headers, and do some tables have header or body cells that span more than one row or column?)

For large, complex tables you will have a lot of work to do no matter which method(s) you choose. Here are some of the considerations:

- Markers have the advantage that displaying them in FrameMaker does not cause a table to balloon into something monstrous and unwieldy. On the other hand, markers can be difficult to work with because of the tiny dialog FrameMaker provides, and because each marker is limited to 256 characters.
- Applying conditional text to special paragraphs containing attribute values, or to extra paragraphs in formats to which attributes are assigned, might seem easier than using markers. However, applying conditions to text in individual cells in FrameMaker tables can be problematic. And when you **Show All**, many of your tables might become unreadable.
- A different paragraph format for the text in each group of cells that needs a particular combination of WAI attributes can work well if your document contains just one large table; but might become a major annoyance if your document contains a lot of tables. Each of those paragraph formats must be unique in your document, and all of them have to be in the catalog.

25.4.3 Providing table summary and title information

In this section:

§25.4.3.1 [Using a table attribute for summary or title](#) on page 761

§25.4.3.2 [Using a dedicated format for table summary or title](#) on page 761

§25.4.3.3 [Using a custom marker for table summary or title](#) on page 762

25.4.3.1 Using a table attribute for summary or title

Under [TableAttributes] specify the summary or title attribute for the table's TableID, and give the attribute a value:

```
[TableAttributes]
TableID= summary="Text of summary for this table"
TableID= title="Text of title for this table"
```

For example, for the summary attribute you would specify something like this:

```
[TableAttributes]
aa123456= summary="This is the text of my summary for this table"
```

where aa123456 consists of the FileID from mif2go.ini (see §5.3.4 [Working with Mif2Go FileIDs](#) on page 119) followed by the TableID from FrameMaker (see §24.2.1 [Determining the TableID](#) on page 729).

You could also specify the following:

```
[TableAttributes]
aa123456= title="My Title Attribute" summary="My summary info"
```

The attributes and values must fit all on one line (of any length) in the configuration file. You could specify a macro instead, and use any number of lines:

```
[TableAttributes]
aa123456= <$attr4aa123456>

[Attr4aa123456]
title="I can put as long a title attribute here as I want"
summary="This is my lengthy and informative table summary"
```

The line breaks in the macro are preserved in the HTML output. See §28 [Working with macros](#) on page 787 for more information.

25.4.3.2 Using a dedicated format for table summary or title

You can designate a paragraph format whose content will be the value of the summary attribute for the next table in your document. For example, suppose you use paragraph format *TblSum* for this purpose:

```
[HTMLParaStyles]
; Summary makes current para content into summary for table tag
TblSum=Summary Delete
```

Somewhere in each table (or just before the table) you would insert a *TblSum* paragraph containing the summary for that table. Probably you would make that paragraph conditional so it would not appear in print. The Delete property would exclude the paragraph (as such) from HTML output; HTML source would show the text of the *TblSum* paragraph as the value for the summary attribute of the <table> element.

If some tables in your document have no captions (FrameMaker table titles), you might want to use the HTML title attribute also, to provide a title. Designate a paragraph format whose content will be the title for any table in which (or just before which) you place an instance of the paragraph. For example, suppose you use paragraph format *TblTtl* for this purpose:

```
[HTMLParaStyles]
; TableTitle makes current para content into title attr for table
TblTtl=TableTitle Delete
```

The content of the *TblTtl* paragraph would become the text of the title attribute of the HTML <table> tag.

25.4.3.3 Using a custom marker for table summary or title

You can use a custom marker to provide a summary for a table, and another custom marker to provide a title. Each marker name must start with **Table** and end with the name of the attribute:

TableSummary Summary attribute for table (maximum 256 characters).

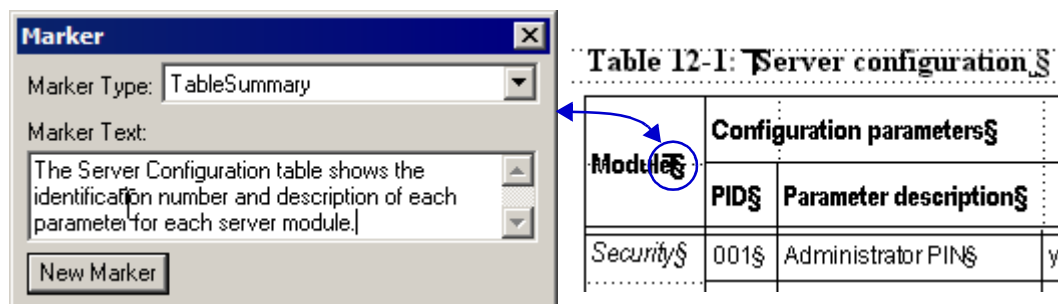
TableTitle Title attribute for a table that has no <caption>.

For example, to add a summary attribute:

1. Create a FrameMaker marker named **TableSummary**.
2. Place a **TableSummary** marker somewhere in the table, or just before the table.
3. Type the content of the summary in the *Marker* dialog, as shown in [Figure 25-1](#).
4. Click **New Marker**.

A **TableSummary** marker placed in the “Server configuration” table shown in [Figure 25-1](#) might have the content shown in the Marker Text box.

Figure 25-1 TableSummary marker



The HTML source would show the <table> tag as follows (omitting display attributes):

```
<table summary="The Server Configuration table shows the
identification number and description for each parameter for each
server module.">
```

Note: If the content of a **TableSummary** or **TableTitle** marker includes characters < or >, these characters must be escaped, thus: \< or \>; otherwise the table might not be rendered correctly in HTML.

25.4.4 Identifying table row and column information

Grouping cells logically for non-visual interpretation, and associating each cell in a table with the actual or virtual header information that informs the content of that cell, can require a lot of markup. Methods for using **Mif2Go** to generate WAI table-cell attributes are described in the following sections:

§26 [Identifying HTML table structure for WAI](#) on page 763

§27 [Marking HTML table cells for WAI](#) on page 775

26 Identifying HTML table structure for WAI

This section describes how to use **Mif2Go** configuration settings to identify table structure for WAI support. Topics include:

§26.1 [Identifying table rows and columns](#) on page 763

§26.2 [Associating table cells with header cells](#) on page 766

See also:

§25 [Generating WAI markup for HTML](#) on page 755

§27 [Marking HTML table cells for WAI](#) on page 775

26.1 Identifying table rows and columns

In this section:

§26.1.1 [Developing a strategy for row and column markup](#) on page 763

§26.1.2 [Comparing scope and id/headers accessibility methods](#) on page 763

§26.1.3 [Specifying a default accessibility method](#) on page 764

§26.1.4 [Overriding the default accessibility method](#) on page 765

26.1.1 Developing a strategy for row and column markup

Mif2Go supports two markup methods for associating table-cell content with row and column header information; you can mix the two approaches:

<code>scope</code>	Indicates the set of body cells to which a header cell applies.
<code>id/headers</code>	Gives each header cell an <code>id=uniqueID</code> attribute. Gives each cell to which that header cell applies a <code>headers=uniqueID</code> attribute.

*Strategy for WAI
table markup*

To keep WAI table markup as simple as possible, consider using this strategy:

1. Decide on a basic policy with respect to accessibility method for all tables in your document, and set `[Tables]AccessMethod` accordingly; see §26.1.3 [Specifying a default accessibility method](#) on page 764.
2. If there are repeating sets of columns or rows in some tables, make the appropriate top/left cells in those tables `ColGroup` or `RowGroup` cells; see §26.2.1 [Specifying group properties for header cells](#) on page 766.
3. Test the result, and if necessary use fine-control settings to correct any problems that might result from complex or unusual table structures; see §26.2 [Associating table cells with header cells](#) on page 766.
4. Add `abbr` and `axis` attributes as needed; see §26.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 768 and §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772.

26.1.2 Comparing scope and id/headers accessibility methods

scope method The `scope` method works well for simple tables. **Mif2Go** places a single attribute in each column-header or row-header cell, to apply that header to the rest of the cells in the same column or row.

<i>id / headers method</i>	The <code>id/headers</code> method is better for complex tables. Mif2Go names each header cell (gives it an <code>id=name</code> attribute) and also indicates that cell's applicability by specifying the <code>headers=name</code> attribute in every affected cell.
<i>Both methods</i>	For some table structures it can also make sense to mix methods, such as using the <code>id/headers</code> method for columns and the <code>scope</code> method for rows.

26.1.3 Specifying a default accessibility method

In this section:

§26.1.3.1 [Establishing a basic policy for table accessibility](#) on page 764

§26.1.3.2 [Applying the scope method to all tables](#) on page 764

§26.1.3.3 [Applying the id/headers method to all tables](#) on page 765

26.1.3.1 Establishing a basic policy for table accessibility

You can set a basic policy for adding accessibility to tables, by specifying a default method for associating table-cell content with row and column header information:

```
[Tables]
; AccessMethod = None (default), Scope, or IDheaders
AccessMethod=None
```

If you specify a default method, **Mif2Go** applies that method to all tables in your document. To specify a different method for selected tables, table formats, or table groups, you can indicate overrides in the `[TableAccess]` section; see §26.1.4 [Overriding the default accessibility method](#) on page 765.

If you specify group properties for some or all header cells (see §26.2.1 [Specifying group properties for header cells](#) on page 766), `AccessMethod` works in concert with these properties.

If you do not specify a default method, or if you specify `AccessMethod=None`, you can still apply one or both methods to all tables by specifying settings for columns, rows, column groups, and row groups; see §27 [Marking HTML table cells for WAI](#) on page 775. This is a good way to mix the two methods, because you can treat columns and rows differently.

Note: If you specify a default method for all tables, do not also use a different setting, or a marker, to apply the *same* method to an individual table; the result is duplicate attribute assignments. See §13.16.5 [Avoiding redundant attribute assignments in tables](#) on page 456.

26.1.3.2 Applying the scope method to all tables

When `AccessMethod=Scope`, **Mif2Go** supplies the following WAI attributes for every table:

- `scope="colgroup"` for any header cells marked `ColGroup`; automatically sets `ColGroupElements=Yes` (see §24.3.2.3 [Enumerating table column groups](#) on page 732) if any column-header cells are so specified.
- `scope="rowgroup"` for any left-side cells marked `RowGroup`; automatically sets `HeadFootBodyTags=Yes` (see §24.3.2.4 [Wrapping table row groups](#) on page 732) if any row-header cells are so specified.
- `scope="row"` or `scope="col"` for the remaining header cells.
- If a header cell spans more than one column or row (and is not marked `ColGroup` or `RowGroup`), it must have an ID even though the method is `scope`, because there is no

WAI attribute for `scope="colspan"` (or `"rowspan"`); such header cells get `id="spanN"` and the cells affected by them get `headers="spanN"`.

In addition, `AccessMethod=Scope` sets [Tables] properties `ScopeColGroup`, `ScopeRowGroup`, `ScopeCol`, and `ScopeRow`; and sets `ColSpanIDs` and `RowSpanIDs` for other straddling header cells. See §27.2 [Using the scope method to identify table cells](#) on page 775 for more information.

Note: If any of your tables have footer rows, when you use the scope method the resulting HTML might contain some surprises; see §24.3.2.4 [Wrapping table row groups](#) on page 732.

26.1.3.3 Applying the id/headers method to all tables

When `AccessMethod=IDheaders`, **Mif2Go** supplies the following WAI attributes for every table:

- `id="groupN"` for any cells marked `ColGroup` or `RowGroup`.
- `id="spanN"` for any spanning cells (and cells marked `Span`).
- `id="rowN"` or `id="colN"` for the remaining cells.
- A `headers` attribute that names all applicable IDs for each affected cell.

In addition, `AccessMethod=IDheaders` sets [Tables] properties `ColGroupIDs`, `RowGroupIDs`, `ColSpanIDs`, `RowSpanIDs`, `ColIDs`, and `RowIDs`, so that most header cells have IDs and the corresponding body cells have matching headers. See §27.3 [Using the id/headers method to identify table cells](#) on page 777 for more information.

26.1.4 Overriding the default accessibility method

You can use settings in the [TableAccess] section to override, for selected tables, the corresponding [Tables] default settings; everything you can set in [TableAccess] has a document-wide default in the [Tables] section.

You can specify overrides that apply to table groups, to tables of a certain FrameMaker format, and to individual tables. You can even use wildcards to specify tables that are not explicitly grouped.

Use these settings to specify accessibility-method overrides:

```
[TableAccess]
; table ID = method list (overrides default in [Tables]); Can
; override [Tables]AccessMethod policy with NoAccess, Scope, or IDs.
```

For example:

```
[TableAccess]
ac254360=NoAccess
Group5=IDs
Format A=HCols1 HRows2 Scope
```

To override the default method for *all* tables for rows, columns, row groups, or column groups, use one of the row or column markup methods instead; see:

§27.2 [Using the scope method to identify table cells](#) on page 775

§27.3 [Using the id/headers method to identify table cells](#) on page 777

To override attributes at the cell level, use one of the table-cell markup methods instead; see:

§26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767

§26.2.3 [Assigning table-cell attribute values with dedicated formats](#) on page 772

§26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772

Note: If you specify a default method for all tables (see §26.1.3 [Specifying a default accessibility method](#) on page 764), do not also use an override to apply the *same* method to an individual table; the result is duplicate attribute assignments. See §13.16.5 [Avoiding redundant attribute assignments in tables](#) on page 456.

26.2 Associating table cells with header cells

If specifying an accessibility method does not prove adequate for some or all tables in your document, **Mif2Go** provides two additional ways to indicate, for WAI purposes, which header cells apply to which other table cells:

- [Tables] settings for rows and columns; these apply by default to all tables, but you can override them with [TableAccess] settings. See §27 [Marking HTML table cells for WAI](#) on page 775 for information about these settings.
- Attributes you specify for rows, columns, or individual cells via paragraph formats or markers; these apply to the tables in which you use them.

In this section:

§26.2.1 [Specifying group properties for header cells](#) on page 766

§26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767

§26.2.3 [Assigning table-cell attribute values with dedicated formats](#) on page 772

§26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772

26.2.1 Specifying group properties for header cells

In this section:

§26.2.1.1 [Defining blocks of header cells](#) on page 766

§26.2.1.2 [Using header cells to define column groups](#) on page 766

§26.2.1.3 [Using header cells to define row groups](#) on page 767

26.2.1.1 Defining blocks of header cells

You can specify that a row- or column-header cell should apply not just to the cells in its immediate row or column, but to a larger group: a block of cells, possibly including other row- or column-header cells. How the group property works depends on whether you are using the `scope` method or the `id/headers` method to associate body cells with header cells.

For example, if you use `scope="colgroup"` in a column-header cell along with `ColGroupElements=Yes`, the table columns to which the header cell applies are all those in its own `<colgroup>` element. If you use `id/headers` instead, the setting for `ColGroupElements` does not matter; **Mif2Go** does the work of making matching `id` and `headers` identifiers. They do the very same job as `scope+ColGroupElements`; the same associations are made from header cells to body cells.

See §27.5 [Using ColGroup and RowGroup cells](#) on page 784 for more information.

26.2.1.2 Using header cells to define column groups

Mif2Go refers to a column-header cell with a group attribute as a *ColGroup cell*. To make a header cell a ColGroup cell, include in it one of the following:

- a paragraph that is in a format you have designated [HTMLParaStyles] ColGroup (see §26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767)

- a **CellGroup** marker that you have given content `col` (see §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772).

When you designate a header cell as a ColGroup cell, the effect of that property depends on which method you specify for table columns:

- `scope` automatically specifies the following:
 - `[Tables]ColGroupElements=Yes`
(ColGroup cell starts a new `<colgroup>` element)
 - `scope="colgroup"` in the ColGroup cell
(ColGroup cell affects all other cells subsumed by its `<colgroup>`)
- `id/headers` automatically specifies the following:
 - `[Tables]ColGroupIDs=Yes`
(ColGroup cell gets `id="groupN"`, dependent cells get `headers="groupN"`)

See §27.5.1 [Understanding how the ColGroup property works](#) on page 784 for more information.

26.2.1.3 Using header cells to define row groups

Mif2Go refers to a row-header cell that has a group attribute as a *RowGroup cell*. To make a header cell a RowGroup cell, include in it one of the following:

- a paragraph that is in a format you have designated `[HTMLParaStyles] RowGroup` (see §26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767)
- a **CellGroup** marker that you have given content `row` (see §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772).

When you designate a header cell as a RowGroup cell, the effect of that property depends on which method you specify for table rows:

- `scope` automatically specifies the following:
 - `HeadFootBodyTags=Yes`
(RowGroup cell starts a new `<tbody>` element)
 - `scope="rowgroup"` in the RowGroup cell
(RowGroup cell affects all other cells in its `<tbody>`)
- `id/headers` automatically specifies the following:
 - `RowGroupIDs=Yes`
(RowGroup cell gets `id="groupN"`, dependent cells get `headers="groupN"`)

See §27.5.2 [Understanding how the RowGroup property works](#) on page 785 for more information.

26.2.2 Using paragraph formats for table-cell attributes

In this section:

- §26.2.2.1 [Choosing how to use paragraph formats for WAI markup](#) on page 767
- §26.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 768
- §26.2.2.3 [Assigning values to WAI attributes](#) on page 769
- §26.2.2.4 [Specifying a different HTML table-cell tag](#) on page 770
- §26.2.2.5 [Identifying table cells with formats: an example](#) on page 770

26.2.2.1 Choosing how to use paragraph formats for WAI markup

To add WAI markup using paragraph formats, you must apply a different paragraph format to the content of each cell that needs a particular combination of WAI markup, in each table. You can apply the unique format to the visible content of the cell; or you can apply it

to extra text in the cell, use property `Delete` to prevent the additional text from appearing in the HTML output, and apply a condition to hide the extra text in print versions. See §25.1.2 [Using paragraph formats for WAI attributes](#) on page 755 for more information.

A paragraph format to which you assign a WAI attribute can be anywhere in the cell, and does not have to be the only paragraph format used in that cell.

26.2.2.2 Assigning WAI attributes to paragraph formats

You can combine format-specific settings in `[HTMLParaStyles]` with attributes you define in other `[HtmlStyle*]` sections to control WAI behavior at the cell level. You must use a different paragraph format for the content of each cell that needs a different set of attributes.

Use these settings to specify WAI attributes for table cells.

```
[HTMLParaStyles]
; format name = properties
; These provide support for Web Accessibility Initiative table markup.
; CellAttribute inserts the contents of [StyleCellAttribute] into
; the start tag of the enclosing table cell (ignored outside tables).
; Span causes assignment of ColSpanID or RowSpanID, as enabled in
; the [Tables] section.
; NoColID prevents assignment of id for ColIDs (enabled in [Tables])
; for any cell that contains an instance of its para format.
; ColGroup is used for para formats in cells in the header row.
; RowGroup is used for para formats in cells at the left of their
; rows.
; Scope looks up value for scope= attribute in [StyleCellScope]
; Abbr looks up value for abbr= attribute in [StyleCellAbbr]
; Axis looks up value for axis= attribute in [StyleCellAxis]
```

[Table 26-1](#) describes each of these properties.

Table 26-1 Format properties for WAI table-cell attributes

Property	Description
Abbr	An abbreviation for the cell's content is assigned to the paragraph format under <code>[StyleCellAbbr]</code> .
Axis	The cell belongs to a category that is not necessarily indicated by the row and column headers with which it is associated; the category (axis) is specified for the paragraph format under <code>[StyleCellAxis]</code> .
CellAttribute	Attributes listed under <code>[StyleCellAttribute]</code> for the paragraph format are applied to the cell. You can list values for other WAI attributes (<code>Scope</code> , <code>Abbr</code> , and <code>Axis</code>) under <code>[StyleCellAttribute]</code> if you want to, instead of listing them in the attribute-specific sections.
ColGroup	The cell is a header cell that starts a column group. See §26.2.1.2 Using header cells to define column groups on page 766.
NoColID	The column to which the cell belongs does not need to be identified for WAI purposes, even though you have specified in the <code>[Tables]</code> section that you want columns identified. This setting is intended to allow skipping columns that are used only for spacing.

Table 26-1 Format properties for WAI table-cell attributes (continued)

Property	Description
RowGroup	The cell is a header cell that starts a row group. See §26.2.1.3 Using header cells to define row groups on page 767.
Scope	The cell is a header cell that applies to a column, a group of columns, a row, or a group of rows, whichever is indicated for the format under [StyleCellScope].
Span	The cell is a header cell that applies to more than one column or row; id="spanN" is assigned to the cell, regardless of ColSpanIDs or RowSpanIDs settings. Often such a header cell actually consists of two or more cells that have been straddled in FrameMaker.

26.2.2.3 Assigning values to WAI attributes

The following configuration-file sections can include settings for WAI attributes that are assigned to paragraph formats in the [HTMLParaStyles] section:

```
[StyleCellAbbr]
; format name = abbr attribute value to insert in enclosing cell

[StyleCellAxis]
; format name = axis attribute value to insert in enclosing cell

[StyleCellScope]
; format name = scope attribute value to insert in enclosing cell,
; required by WAI to be one of col, colgroup, row, or rowgroup.

[StyleCellAttribute]
; doc style = attribute to insert in enclosing table cell start tag,
; used in addition to other cell attributes given under [Table...]
```

You can use the format or an abbreviation of the cell content to specify the scope:

[Format example](#)

[Abbreviation example](#)

Format example

For example, if you are using column groups and a table has a column header that applies to (has a scope of) more than one column, you might give the text of the header a unique paragraph format (such as *WideHdr*), and specify the following settings:

```
[HTMLParaStyles]
WideHdr=Scope

[StyleCellScope]
WideHdr=colgroup
```

Instead of using [StyleCellScope], you could specify the colgroup attribute like this:

```
[HTMLParaStyles]
WideHdr=CellAttribute

[StyleCellAttribute]
WideHdr= scope="colgroup"
```

Abbreviation example

Suppose the header-cell content with paragraph format *WideHdr* is “Type of convention”. To abbreviate this text to “Type”, you could specify both attributes like this:

```
[HTMLParaStyles]
WideHdr=Scope CellAttribute

[StyleCellAttribute]
WideHdr= abbr="Type" scope="colgroup"
```

or like this:

```
[HTMLParaStyles]
WideHdr=Scope Abbr
```

```
[StyleCellScope]
WideHdr=colgroup
[StyleCellAbbr]
WideHdr="Type"
```

26.2.2.4 Specifying a different HTML table-cell tag

You can use the following settings to alter the HTML tag for table cells containing the designated paragraph formats:

```
[HTMLParaStyles]
; doc style (para or char) = keywords for functions and properties
; These alter properties or attributes of their table cell:
; TableHead forces containing cell tag to th instead of td
; TableBody forces containing cell tag to td instead of th
CellHeadParaFormat=TableHead
CellBodyParaFormat=TableBody
```

You might want to do this if some tables in your document have a structure different from that described by document-wide [Tables] settings. A more straightforward method is to use [TableAccess] settings to override the [Tables] settings for selected tables, and thus avoid dedicating a paragraph format to this purpose. See §24.3.3 [Identifying table headers and footers](#) on page 734.

26.2.2.5 Identifying table cells with formats: an example

Suppose your FrameMaker document contains a table with the following structure; [Table 26-2](#) has these characteristics:

- multiple header rows
- a column-header cell that spans more than one column
- a row-header column whose cells span more than one row
- a column that has no header.

Table 26-2 Using paragraph formats to identify table cells (example)

Configuration parameters				<< Column-header rows have paragraph format CellHead except top row, which has CellHeadM.
Module	PID	Parameter description		
Security	001	Administrator PIN	y	<< Body cells have paragraph format CellBody, except cells in rightmost column, which have CellBodyN.
	012	Private key	y	
Certificate	002	Authority certificate	n	
	011	Manager certificate	n	
	009	Server certificate	y	

^^ Paragraph format is CategoryM for body cells in the first column = row headers.

To use paragraph formats to identify body cells according to their row headers and column headers, those header cells that span more than one row or column must contain a paragraph format different from (or perhaps in addition to) the paragraph format used in ordinary row and column headers:

- Because it spans more than one column, the topmost column-header cell in [Table 26-2](#) needs special identification, so a different paragraph format is used for that cell.
- All row-header cells span more than one row, so no individual row-header cell needs a format different from any other. However, collectively the row-header contents need a paragraph format different from the format for column headers.
- The rightmost column in [Table 26-2](#) has no header; a different paragraph format is used to identify the cells in this column, in order to give them the NoColID attribute.

You could specify the following attributes for [Table 26-2](#):

```
[Tables]
UseTbHeaderCode=Yes
TableHeaderCols=1
ColIDs=Yes
ColHead=col
ColSpanIDs=Yes
ColSpanHead=span
RowIDs=Yes
RowHead=row
RowSpanIDs=Yes
RowSpanHead=span
```

Because these settings specify enough information to associate every cell in the table with all applicable row and column headers, there is no need for the `Scope` attribute. However, using it does no harm, so `Scope` is included for purposes of illustration:

```
[HTMLParaStyles]
CellHead=Scope
CellHeadM=Scope Span
CategoryM=Scope Span
CellBodyN=NoColID

[StyleCellScope]
CellHead=column
CellHeadM=colgroup
CategoryM=rowgroup
```

Because `ColIDs` take precedence over `RowIDs`, the top left cell gets `id="col1"`. The cell to its right is in column 2; the cell below it is in row 2. [Table 26-2](#) on page 770 looks something like this (omitting display attributes) in **Mif2Go**-generated HTML:

```
<table>
<caption><p>Table 26-2: Server configuration</p></caption>
<tr><th id="col1" scope="column" rowspan="2"><p>Module</p></th>
  <th id="span1" scope="colgroup" colspan="3">
    <p>Configuration parameters</p></th></tr>
<tr><th id="col2" scope="column" headers="span1">
  <p>PID</p></th>
  <th id="span2" scope="column" colspan="2">
    <p>Parameter description</p></th></tr>
<tr><th id="span3" scope="rowgroup" headers="col1" rowspan="2">
  <p>Security</p></th>
  <td id="row3" headers="col2 span1 span3"><p>001</p></td>
  <td headers="col2 row3 span1 span2 span3">
    <p>Administrator PIN</p></td>
  <td headers="row3 span1 span2 span3"><p>y</p></td></tr>
<tr><td id="row4" headers="col2 span1 span3"><p>012</p></td>
  <td headers="col2 row4 span1 span2 span3"><p>Private key</p></td>
  <td headers="row4 span1 span2 span3"><p>y</p></td></tr>
<tr><th id="span4" scope="rowgroup" headers="col1" rowspan="3">
  <p>Certificate</p></th>
  <td id="row5" headers="col2 span1 span4"><p>002</p></td>
  <td headers="col2 row5 span1 span2 span4">
    <p>Authority certificate</p></td>
  <td headers="row5 span1 span2 span4"><p>n</p></td></tr>
<tr><td id="row6" headers="col2 span1 span4"><p>011</p></td>
  <td headers="col2 row6 span1 span2 span4">
    <p>Manager certificate</p></td>
  <td headers="row6 span1 span2 span4"><p>n</p></td></tr>
<tr><td id="row7" headers="col2 span1 span4"><p>009</p></td>
  <td headers="col2 row7 span1 span2 span4">
    <p>Server certificate</p></td>
```



```
<td headers="row7 span1 span2 span4"><p>y</p></td></tr>
</table>
```

26.2.3 Assigning table-cell attribute values with dedicated formats

Instead of inventing another paragraph format every time you need to assign a different combination of WAI attributes, you can dedicate a small set of paragraph formats to this purpose: one for each WAI attribute. The text of each instance of such a paragraph format becomes the value of the attribute:

```
[HTMLParaStyles]
; These para format properties all make their content into attributes.
; If you do not want the content in the text also, use with Delete.
; AbbrVal makes current para content into abbr for table cell
; AxisVal makes current para content into axis for table cell
```

Probably you would not want the text of these special paragraphs to appear either in printed output or in HTML output; therefore you would assign property `Delete` to each such paragraph format in section `[HTMLParaStyles]`, and in `FrameMaker` make the paragraphs conditional, so you can hide them.

For example, suppose you add paragraph format *WAIabbr* to the paragraph catalog, and assign a WAI attribute to this format:

```
[HTMLParaStyles]
WAIabbr = AbbrVal Delete
```

If a header cell in a table reads *Type of Widget* and you want to provide the abbreviation *Type*, somewhere in that cell you would place a *WAIabbr* paragraph and give it content *Type*. You would make the *WAIabbr* paragraph conditional so it would not appear in print. The `Delete` property would exclude the paragraph (as such) from HTML output, and the HTML source would show `<abbr="Type">` for the cell in question; see §21.3.12 [Eliminating unwanted paragraphs](#) on page 652.

26.2.4 Assigning table-cell attribute values with custom markers

You can use special `FrameMaker` markers to apply WAI cell attributes. Each marker name must start with **Cell** and end with the name of an attribute. The markers for table cells are as follows:

CellAbbr	Abbreviation for content of a cell; adds <code>abbr</code> attribute.
CellAxis	Conceptual category for the content of a cell; adds <code>axis</code> attribute.
CellID	Cell identifier; replaces the value of any generated <code>id</code> attribute.
CellScope	Number of rows or columns covered by a header cell; adds <code>scope</code> attribute.

For example, to add the `abbr` attribute to a table cell:

1. Create a `FrameMaker` marker type named **CellAbbr**.
2. For each cell whose content needs an abbreviation:
 - 2.1. Place a **CellAbbr** marker in the cell.
 - 2.2. Type the abbreviation in the *Marker* dialog.
 - 2.3. Click **New Marker**.

Two additional marker types do not conform to the naming convention described above. Instead, they provide the same effects as certain properties assigned to paragraph formats in the [HTMLParaStyles] section:

- | | |
|------------------|---|
| CellGroup | The marker text must contain either <code>col</code> or <code>row</code> ; the effect is as though <code>ColGroup</code> or <code>RowGroup</code> was assigned to a paragraph format in the cell. See §26.2.1 Specifying group properties for header cells on page 766. |
| CellSpan | The marker text can contain anything, but must not be empty; the effect is as though <code>Span</code> was assigned to a paragraph format in the cell. |

For information about assigning properties `ColGroup`, `RowGroup`, and `Span`, see §26.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 768.

Note: If you specify a default access method for all tables (see §26.1.3 [Specifying a default accessibility method](#) on page 764), do not also use a marker to apply the *same* method to individual tables; the result is duplicate attribute assignments. See §13.16.5 [Avoiding redundant attribute assignments in tables](#) on page 456.

27 Marking HTML table cells for WAI

This section describes how to use **Mif2Go** configuration settings to fine-tune the association of table-cell content with row- and column-header information. Topics include:

§27.1 [Understanding table cell settings](#) on page 775

§27.2 [Using the scope method to identify table cells](#) on page 775

§27.3 [Using the id/headers method to identify table cells](#) on page 777

§27.4 [Overriding default table-cell settings](#) on page 784

§27.5 [Using ColGroup and RowGroup cells](#) on page 784

See also:

§25 [Generating WAI markup for HTML](#) on page 755

§26 [Identifying HTML table structure for WAI](#) on page 763

27.1 Understanding table cell settings

You use [Tables] settings to specify WAI attributes that associate table cells with rows, row groups, columns, and column groups. *These settings apply to all tables in your document.* You can use corresponding [TableAccess] settings to override many of them for selected tables. To specify different [Tables] settings for all the tables in a single FrameMaker chapter file, you can include those settings in a configuration file named after the chapter file; for example, Chap2.ini. See §33.1 [Using a different configuration for selected files](#) on page 919.

Mif2Go generates identifiers for each cell from the [Tables] settings, to associate the cell with the specified parts of the table. To avoid duplicate cell identifiers when an output file includes more than one table, **Mif2Go** adds to each identifier a string that is unique to each table in the file. For example, all identifiers in the first table in the file end with t1, those in the next table end with t2, and so forth.

27.2 Using the scope method to identify table cells

[Table 27-1](#) lists the scope settings you can specify in the [Tables] section of the configuration file.

Table 27-1 WAI scope attributes for table cells

	[Tables] setting	Default value	[TableAccess] override	Purpose
Column	ScopeCol	No	ScopeCol	Apply scope="col" (the default) to column-header cells
	ScopeColGroup	No	ScopeColGroup	Apply scope="colgroup" to ColGroup header cells*
Row	ScopeRow	No	ScopeRow	Apply scope="row" (the default) to row-header cells
	ScopeRowGroup	No	ScopeRowGroup	Apply scope="rowgroup" to RowGroup header cells*

* Cells marked as ColGroup or RowGroup via [HTMLParaStyles]parafmt=*Group or CellGroup marker

Use these settings to identify column and row header cells that apply to more than one body column or row, either explicitly (via straddles in FrameMaker) or implicitly:

```
[Tables]
; ScopeCol = No (to not use) or Yes (to apply default scope="col"
; to non-empty cells in table header)
ScopeCol=No
; ScopeColGroup = No (to not use) or Yes (to apply scope="colgroup"
; instead of "col" to column head cells identified as ColGroup via
; [HTMLParaStyles] or CellGroup marker col; sets ColGroupElements).
ScopeColGroup=No
; ScopeRow = No (to not use) or Yes (to apply default scope="row"
; to first non-empty cell in each row in the table)
ScopeRow=No
; ScopeRowGroup = No (to not use) or Yes (to apply scope="rowgroup"
; instead of "row" to non-empty row-spanning cells at left in table;
; applies "row" to non-spanning cells, so ScopeRow is not needed).
ScopeRowGroup=No
```

You can override each of these settings in the [TableAccess] section for selected tables by specifying the same setting, prefixed with No, as a property; see §27.4 [Overriding default table-cell settings](#) on page 784.

Note: If you set AccessMethod=Scope, Mif2Go automatically sets ScopeCol, ScopeRow, ScopeColGroup, and ScopeRowGroup to Yes.

Columns and rows

ScopeCol applies to non-empty cells in rows that are tagged <th> or that are designated as header rows via [Tables]TableHeaderRows or [TableAccess]HRowsN.

ScopeRow applies to non-empty cells in the first (leftmost) column in the table, even if the cells in that column are tagged <td> instead of <th>; or to columns that are designated as row headers via [Tables]TableHeaderCols or [TableAccess]HColsN.

Groups of columns or rows

You can use scope=colgroup or scope=rowgroup to apply a header to all cells in a group. If you use column groups and row groups, you can specify a group scope even though none of the header cells spans more than one column or row.

For the group scope settings to be meaningful and effective, a table has to have the structure they imply. For example, scope="colgroup" works only if the table has column groups (<colgroup> elements), and scope="rowgroup" works only if the table has row groups (<tbody> elements). Therefore:

- Specifying column groups automatically sets [Tables]ColGroupElements=Yes; for more information, see §24.3.2.3 [Enumerating table column groups](#) on page 732.
- Specifying row groups automatically sets [Tables]HeadFootBodyTags=Yes; for more information, see §24.3.2.4 [Wrapping table row groups](#) on page 732.

The group scope attributes work in concert with ColGroup and RowGroup cells: header cells that are assigned [HTMLParaStyles] property ColGroup or RowGroup, described in §26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767; or that contain marker type **CellGroup**, described in §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772.

Note: If any of your tables have footer rows, when you use scope="rowgroup" the resulting HTML might contain some surprises; see §24.3.2.5 [Positioning table footer rows \(deprecated\)](#) on page 733 in §24.3.2.4 [Wrapping table row groups](#) on page 732.

27.3 Using the id/headers method to identify table cells

In this section:

- §27.3.1 [Choosing an id/headers level](#) on page 777
- §27.3.2 [Specifying id/headers attributes for table cells](#) on page 777
- §27.3.3 [Grouping header cells for identification](#) on page 778
- §27.3.4 [Column-group and row-group extent](#) on page 779
- §27.3.5 [Choosing a different row-group method](#) on page 780
- §27.3.6 [Using span attributes to identify rows and columns](#) on page 780
- §27.3.7 [Column-span and row-span extent](#) on page 781
- §27.3.8 [Identifying individual table cells by row and column](#) on page 782
- §27.3.9 [Column and row extent](#) on page 783
- §27.3.10 [Using span IDs with row or column IDs](#) on page 783

27.3.1 Choosing an id/headers level

If you decide to use the `id/headers` method, you can choose from three levels:

- Groups:* Identify column-header cells or row-header cells that apply to a block of cells, including other header cells; add `headers` attributes to all affected cells, identifying each by the header cell of its block.
- Spans:* Identify column-header or row-header cells that explicitly or implicitly apply to multiple columns or rows of body cells; add `headers` attributes to all affected body cells, identifying each by the header cells that apply.
- Cells:* Identify each column-header cell and row-header cell; add `headers` attributes to all body cells, identifying each by row and column.

First see if you can use groups to adequately identify cells; if grouping header cells does not give you enough resolution, consider span attributes; if span attributes do not suffice, use row and column IDs to provide the maximum amount of identification for each cell.

If you need to identify cells by virtual or conceptual properties, or by disjoint groupings of header cells, you might want to apply the `axis` attribute also, using one of the table markup methods described in §26.2.3 [Assigning table-cell attribute values with dedicated formats](#) on page 772 or §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772.

27.3.2 Specifying id/headers attributes for table cells

[Table 27-2](#) shows the `id/headers` attributes you can specify in the `[Tables]` section. For selected tables you can override each of the `*IDs` settings in the `[TableAccess]` section, by specifying the same setting, prefixed with `No`, as a property; see §27.4 [Overriding default table-cell settings](#) on page 784.

Table 27-2 WAI id/header table cell attributes

	[Tables] setting	Default value	Purpose	Ref.
Column	ColGroupHead	group	Name the id to use for column group headers	27.3.3
	ColGroupIDs	No	Add id="groupN" to ColGroup* header cells, headers="groupN" to cells in the column group	
	ColSpanHead	span	Name the id to use for column-spanning headers	27.3.6
	ColSpanIDs	No	Add id="spanN" to column-header cells designated Span via [HTMLParaStyles] or CellSpan marker, headers="spanN" to cells in the column	
	ColHead	col	Name the id to use for columns	27.3.8
	ColIDs	No	Add id="colN" to column headers, headers="colN" to cells in the column	
Row	RowGroupHead	group	Name the id to use for row group headers	27.3.3
	RowGroupIDs	No	Add id="groupN" to RowGroup* header cells, headers="groupN" to cells in the row group	
	RowSpanHead	span	Name the id to use for row-spanning headers	27.3.6
	RowSpanIDs	No	Add id="spanN" to row-header cells (first cell in each row) designated Span via [HTMLParaStyles] (or via a CellSpan marker), headers="spanN" to cells in the row	
	RowHead	row	Name the id to use for rows	27.3.8
	RowIDs	No	Add id="rowN" to leftmost column, headers="rowN" to cells in the row	
* Cells marked ColGroup or RowGroup via [HTMLParaStyles]parafmt=xGroup or via CellGroup marker				

27.3.3 Grouping header cells for identification

Use the following settings to specify whether header cells should be grouped. These settings work in concert with ColGroup and RowGroup cells: header cells that are assigned [HTMLParaStyles] property ColGroup or RowGroup, described in §26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767; or that contain marker type **CellGroup**, described in §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772.

```
[Tables]
; ColGroupHead is "group" by default; it is used in the id attrs of
; header cells containing a para format with [HTMLParaStyles]ColGroup.
ColGroupHead=group
; RowGroupHead is "group" by default; it is used in the id attrs of
; left cells containing a para format with [HTMLParaStyles]RowGroup.
; If ColGroup is used, first ID numerically follows last ColGroup ID.
RowGroupHead=group
; ColGroupIDs = No (default)
; or Yes (to use id="groupN" in col head cells identified
; as ColGroup via [HTMLParaStyles] or the CellGroup marker col.)
ColGroupIDs=No
```



```

; RowGroupIDs = No (default)
; or Yes (to use id="groupN" in row head cells identified
; as RowGroup via [HTMLParaStyles] or the CellGroup marker row.)
RowGroupIDs=No

```

Column-group and row-group identifiers

The values you specify for ColGroupHead and RowGroupHead are the names **Mif2Go** uses for column-group and row-group identifiers. For example, if you specify ColGroupHead=gname, every ColGroup cell gets attribute id="gnameN". If you do not specify values for ColGroupHead and RowGroupHead, **Mif2Go** uses the default name, group, for both; and numbers the row groups starting where the column-group numbers end.

Column groups

When you specify ColGroupIDs=Yes, **Mif2Go** generates the following identifiers:

```
id="groupN"
```

- for each ColGroup cell (header cell that contains either a paragraph designated [HTMLParaStyles] ColGroup, or a **CellGroup** marker with content col).

```
headers="groupN"
```

- for each cell to the right of the id="groupN" cell until the next ColGroup cell;
- for each cell below the id="groupN" cell;
- for each cell below the headers="groupN" cells that are in the id=groupn row.

Specifying ColGroupIDs=Yes prevents assignment of a single-column ID to the id="groupN" cell, but does not prevent this assignment to the headers=groupN cells to the right of the id="groupN" cell. See §27.3.8 [Identifying individual table cells by row and column](#) on page 782 for information about specifying IDs for individual columns. See also §27.5.1 [Understanding how the ColGroup property works](#) on page 784.

Row groups

When you specify RowGroupIDs=Yes, **Mif2Go** generates the following identifiers:

```
id="groupN"
```

- for each RowGroup cell (header cell that contains either a paragraph designated [HTMLParaStyles] RowGroup, or a **CellGroup** marker with content row) and that does not have a column ID.

```
headers="groupN"
```

- for each cell below the id="groupN" cell until the next RowGroup cell;
- for each cell to the right of the id="groupN" cell;
- for each cell to the right of the headers="groupN" cells that are in the id="groupN" column.

Specifying RowGroupIDs=Yes prevents assignment of a row ID to the RowGroup cell, but does not prevent this assignment to the headers="groupN" cells below the id="groupN" cell. See §27.3.8 [Identifying individual table cells by row and column](#) on page 782 for information about specifying IDs for individual rows. See also §27.5.2 [Understanding how the RowGroup property works](#) on page 785.

27.3.4 Column-group and row-group extent

Figure 27-1 shows the range of cells that are given headers="colgrp1" or headers="rowgrp2", or both, when ColGroupIDs=Yes, RowGroupIDs=Yes, ColGroupHead=colgrp, and RowGroupHead=rowgrp.

Figure 27-1 Extent of row and column groups

	id=colgrp1		id=colgrp2	
id=rowgrp1				
id=rowgrp2				
id=rowgrp3				

headers=colgrp1

headers=rowgrp2

headers=colgrp1 rowgrp2

27.3.5 Choosing a different row-group method

If you set `HeadFootBodyTags=Yes`, probably you will not want to use `RowGroupIDs`; use the `scope` method instead. For example, if you use paragraph format *RowGroupHeading* for row-group header text, you could specify:

```
[HTMLParaStyles]
RowGroupHeading=RowGroup Scope TableHead

[StyleCellScope]
RowGroupHeading=rowgroup
```

or, if the *RowGroupHeading* cells actually span the rows in the group:

```
[Tables]
ScopeRowGroup=Yes
```

either of which produces:

```
<tbody>
<tr><th scope=rowgroup>My Group Head</th><td></td> ... </tr>
<tr><td></td>... </tr>
...
</tbody>
```

This provides the association between “My Group Head” and all the cells in the `<tbody>` section at minimum cost in HTML coding and file size. Only if you cannot use `HeadFootBodyTags`, perhaps because your target browser does not support it, would you want to use `RowGroupIDs` for this purpose.

27.3.6 Using span attributes to identify rows and columns

If complex tables contain header cells that span more than one column or row, you can use the following settings to have **Mif2Go** generate span-numbered `id` attributes for the dependent cells. These settings work in concert with `Span` cells: header cells that are assigned `[HTMLParaStyles]` property `Span`, described in §26.2.2 [Using paragraph formats for table-cell attributes](#) on page 767; or that contain marker type **CellSpan**, described in §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772.

```
[Tables]
; ColSpanIDs = No (to use only per markers or formats), or Yes
; adds id=spanN to each cell in header rows that spans columns,
; or that has a CellSpan marker, or contains any para formats
; with [HTMLParaStyles] Span, increments for each one used.
; adds headers=spanN to all cells below the spanning cell.
ColSpanIDs=No
; ColSpanHead is usually "span".
ColSpanHead=span
; RowSpanIDs = No (to use only per markers or formats), or Yes
; adds id=spanN to first cell in each row if it spans rows, or
```

```

;   if it has a ColSpan marker, or if it has any para formats
;   with [HTMLParaStyles] Span, increments for each one used.
;   adds headers=spanN to all cells right of the spanning cell.
;   if ColSpan used, first ID numerically follows last ColSpanID.
RowSpanIDs=No
; RowSpanHead is usually also "span"; that's why the ID numbers
; used for ColSpan are skipped for RowSpan
RowSpanHead=span

```

Mif2Go implements cell spans so that you can have several span values that all apply to the same cell. If you specify the [HTMLParaStyles] Span property for paragraph formats (or insert **CellSpan** markers) in multiple header columns or rows, and the higher-level headers really do span the columns or rows they affect, their span values appear in each dependent cell's attributes.

You can override each of the *IDs settings in the [TableAccess] section for selected tables by specifying the same setting, prefixed with No, as a property; see §27.4 [Overriding default table-cell settings](#) on page 784.

Column-span and row-span identifiers

The values you specify for ColSpanHead and RowSpanHead are the names **Mif2Go** uses for column-spanning and row-spanning header-cell identifiers. For example, if you specify ColSpanHead=sname, every column-header Span cell gets attribute id="snameN". If you do not specify values for ColSpanHead and RowSpanHead, **Mif2Go** uses the default, span, for both; and numbers the row-spanning header cells starting where the column-spanning numbers end.

Column spans

When you specify ColSpanIDs=Yes, **Mif2Go** generates the following identifiers:

id="spanN"	for each column-header Span cell (cell containing a paragraph designated [HTMLParaStyles] Span, or a CellSpan marker).
headers="spanN"	for each cell in each column below (spanned by) the id="spann" cell.

Row spans

When you specify RowSpanIDs=Yes, **Mif2Go** generates the following identifiers:

id="spanN"	for each row-header Span cell (cell containing a paragraph designated [HTMLParaStyles] Span, or a CellSpan marker), and that does not have a column ID.
headers="spanN"	for each cell in each row to the right of the id="spann" cell.

27.3.7 Column-span and row-span extent

[Figure 27-2](#) shows the range of cells that are given headers="cspan1" or headers="rspan2", or both, when ColSpanIDs=Yes, RowSpanIDs=Yes, ColSpanHead=cspan, and RowSpanHead=rspan.

Figure 27-2 Extent of column and row spans

	id=cspan1		
id=rspan1			
id=rspan2			

headers=cspan1

headers=rspan2

headers=cspan1 rspan2

27.3.8 Identifying individual table cells by row and column

When you use the following settings, **Mif2Go** automatically generates the WAI `id` attribute for all row and column headers, and the `headers` attribute for all individual cells:

```
[Tables]
;RowIDs and ColIDs set row and col IDs in table header cells
; and the matching headers attribute in table body cells.
; ColIDs = No (to not use), or Yes
; adds id=colN to first cell in header row of each column,
; adds headers=colN to each cell below in the same column(s).
ColIDs=No
; ColHead is often seen in examples as "header", but this is
; not essential; it can be any useful identifier:
ColHead=col
; RowIDs = No (to not use), or Yes
; if ColIDs are used, does not affect all the header rows.
; adds id=rowN attribute to the first cell of each body row,
; adds headers=rowN to each following cell in that row.
RowIDs=No
; RowHead is usually row, but again could be anything:
RowHead=row
```

You can override each of the *IDs settings in the [TableAccess] section for selected tables by specifying the same setting, prefixed with `No`, as a property; see §27.4 [Overriding default table-cell settings](#) on page 784.

Column and row identifiers

The values you specify for `ColHead` and `RowHead` are the names **Mif2Go** uses for column and row identifiers. For example, if you specify `ColHead=name`, every cell in column `N` gets attribute `id="nameN"`. If you do not specify values for `ColHead` and `RowHead`, **Mif2Go** uses the defaults: `col` and `row`, respectively.

Columns

When you specify `ColIDs=Yes`, **Mif2Go** generates the following identifiers for each column:

```
id="colN"           for the first (top left) header cell in column n
headers="colN"      for each body cell in column n
```

Mif2Go interprets straddled column-header cells as applying to all the body cells under them. For example, if the header cell for column 3 also straddles columns 4 and 5, **Mif2Go** generates `headers="col3"` for the body cells in columns 3, 4, and 5.



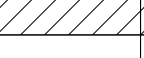
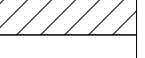
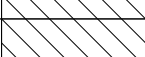
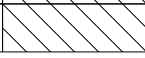

Rows When you specify RowIDs=Yes, **Mif2Go** generates the following identifiers for each row:




<code>id="rowN"</code>	for the first (top left) cell in row <i>n</i> that does not already contain an <code>id</code> attribute (such as <code>id="col1"</code> in the first header row)
<code>headers="rowN"</code>	for each body cell in row <i>n</i> to the right of the <code>id=rowN</code> cell

27.3.9 Column and row extent

Figure 27-3 shows the range of cells that are given a `headers="col2"` attribute, a `headers="row3"` attribute, or both, when ColIDs=Yes, RowIDs=Yes, ColHead=col, and RowHead=row.

Figure 27-3 Extent of column and row IDs

	id=col2	id=col3	id=col4
id=row2			
id=row3			
id=row4			
id=row5			
id=row6			

 `headers=col2`
 `headers=row3`
 `headers=col2 row3`

27.3.10 Using span IDs with row or column IDs

When ColIDs=Yes:

- If ColSpanIDs=No, **Mif2Go** interprets horizontally straddled cells in a column-header row as applying to all the body cells below them. For example, if the first cell in column 2 also straddles the cell next to it in column 3, **Mif2Go** generates `headers="col2"` for the body cells in columns 2 and 3.
- If ColSpanIDs=Yes, the cells are identified as follows:
 - The straddling cell gets `id="spann"` instead of `id="col2"`.
 - The two non-straddling cells in the first row below the straddling cell get `id="col2"` (left cell) and `id="col3"` (right cell).
 - The rest of the non-straddling cells below get `headers="col2"` (left column) and `headers="col3"` (right column).
 - All cells below the straddling cell get `headers="spann"`.

When RowIDs=Yes:

- If RowSpanIDs=No, **Mif2Go** interprets vertically straddled cells in a row-header column as applying to all the body cells to the right of them. For example, if the first cell in row 2 also straddles the cell below it in row 3, **Mif2Go** generates `headers="row2"` for the body cells in rows 2 and 3.
- If RowSpanIDs=Yes, the cells are identified as follows:
 - The straddling cell gets `id="spann"` instead of `id="row2"`.
 - The two non-straddling cells in the first column to the right of the straddling cell get `id="row2"` (top cell) and `id="row3"` (bottom cell).
 - The rest of the non-straddling cells to the right get `headers="row2"` (top row) and `headers="row3"` (bottom row).
 - All cells to the right of the straddling cell get `headers="spann"`.

See §27.3.6 [Using span attributes to identify rows and columns](#) on page 780 for more information about RowSpanIDs.

27.4 Overriding default table-cell settings

You can use settings in the [TableAccess] section to override, for selected tables, the corresponding [Tables] default settings; everything you can set in [TableAccess] has a document-wide default in the [Tables] section.

You can specify overrides that apply to table groups, to tables of a certain FrameMaker format, and to individual tables. You can even use wildcards to specify tables that are not explicitly grouped.

You can negate a default setting by prefixing any of the following keywords with No:

ColIDs
ColGroupIDs
ColSpanIDs
RowIDs
RowGroupIDs
RowSpanIDs
ScopeCol
ScopeColGroup
ScopeRow
ScopeRowGroup

For example:

```
[TableAccess]
; table ID = method list (overrides default in [Tables]) of
; ColIDs, RowIDs, ColSpanIDs, RowSpanIDs, ScopeCol, ScopeColGroup,
; ScopeRow, ScopeRowGroup, ColGroupIDs, RowGroupIDs,
; or any prefixed with No, such as NoColIDs.
aal23456=ColIDs NoRowIDs
group5=ScopeColGroup
Format A=RowSpanIDs NoColIDs
```

27.5 Using ColGroup and RowGroup cells

ColGroup and RowGroup designations describe a structural fact about a table: that the contents of the column or row header cell applies beyond its own column or row.

In this section:

§27.5.1 [Understanding how the ColGroup property works](#) on page 784

§27.5.2 [Understanding how the RowGroup property works](#) on page 785

See §26.2.1 [Specifying group properties for header cells](#) on page 766 for information about specifying ColGroup and RowGroup cells.

27.5.1 Understanding how the ColGroup property works

When you designate a header cell as a ColGroup cell, the effect of that property on the table depends on which accessibility method you have specified:

scope	(via AccessMethod=Scope or via ScopeColGroup=Yes)
id/headers	(via AccessMethod=IDheaders or via ColGroupIDs=Yes)

Using the `scope` method automatically specifies `ColGroupElements=Yes`; the `ColGroup` cell starts a new `<colgroup>` element; and the `ColGroup` cell's information applies to all cells subsumed by that element.

`ColGroupElements=Yes` is a necessary condition for `scope="colgroup"`, but not for `id/headers="groupN"`; for the latter, the `ColGroupElements` value does not affect which cells are marked `id/headers="groupN"`.

`ColGroupIDs=Yes` is a necessary condition for `id/headers="groupN"`, but not for `scope="colgroup"`; for the latter, the `ColGroupIDs` value does not affect which cells are subsumed under `scope="colgroup"`.

Table 27-3 summarizes the effects of the `ColGroup` property when combined with these settings.

Table 27-3 ColGroup property effects

Setting	ColGroupElements	Yes				No			
	ColGroupIDs	Yes		No		Yes		No	
	<code>scope="colgroup" *</code>	Yes	No	Yes	No	Yes	No	Yes	No
Effect	starts new <code><colgroup></code>	Yes	Yes	Yes	Yes	No	No	No	No
	<code>id/headers="groupN"</code>	Yes	Yes	No	No	Yes	Yes	No	No
	<code>scope</code> attribute applied	Yes	No	Yes	No	No	No	No	No

* Set via **CellScope** marker or `[HTMLParaStyles]fmt=Scope`, `[HtmlStyleCellScope]fmt=colgroup`

If `ColGroupElements=Yes`, each `ColGroup` cell starts a new `<colgroup>` element. If the `ColGroup` cell contains a **CellScope** marker (or the `[HTMLParaStyles]/[StyleCellScope]` equivalent) that sets the `scope="colgroup"` attribute, the `ColGroup` property works in concert with the `scope` attribute to apply the `ColGroup` header to all cells subsumed by its `<colgroup>`. The `scope` attribute is in effect only within the same `<colgroup>` section as the `ColGroup` cell. See §27.2 [Using the scope method to identify table cells](#) on page 775.

If `ColGroupIDs=Yes`, each `ColGroup` cell gets an `id="groupN"` attribute; cells below the header cell and to the right of the header-cell column, across to the next `ColGroup` header cell or to the edge of the table (see [Figure 27-1](#) on page 780), each get a matching `headers="groupN"` attribute. If `ColGroupElements=Yes`, these are the cells subsumed by the `<colgroup>` element. See §27.3.3 [Grouping header cells for identification](#) on page 778.

27.5.2 Understanding how the RowGroup property works

When you designate a header cell as a `RowGroup` cell, the effect of that property depends on which of the following you specify also:

`scope` (via `AccessMethod=Scope` or via `ScopeRowGroup=Yes`)

`id/headers` (via `AccessMethod=IDheaders` or via `RowGroupIDs=Yes`)

Using the `scope` method automatically specifies `HeadFootBodyTags=Yes`; the `RowGroup` cell starts a new `<tbody>` element; and the `RowGroup` cell's information applies to all cells in that element.

`HeadFootBodyTags=Yes` is a necessary condition for `scope="rowgroup"`, but not for `id/headers="groupN"`; for the latter, the `HeadFootBodyTags` value does not affect which cells are marked `id/headers="groupN"`.

RowGroupIDs=Yes is a necessary condition for `id/headers="groupN"`, but not for `scope="rowgroup"`; for the latter, the RowGroupIDs value does not affect which cells are subsumed under `scope="rowgroup"`.

Table 27-4 summarizes the effects of the RowGroup property when combined with these settings.

Table 27-4 RowGroup property effects

Setting	HeadFootBodyTags	Yes				No			
	RowGroupIDs	Yes		No		Yes		No	
	<code>scope="rowgroup" *</code>	Yes	No	Yes	No	Yes	No	Yes	No
Effect	starts new <tbody>	Yes	Yes	Yes	Yes	No	No	No	No
	<code>id/headers="groupN"</code>	Yes	Yes	No	No	Yes	Yes	No	No
	scope attribute applied	Yes	No	Yes	No	No	No	No	No

* Set via **CellScope** marker or `[HTMLParaStyles]fmt=Scope, [StyleCellScope]fmt=rowgroup`

If `HeadFootBodyTags=Yes`, each RowGroup cell starts a new <tbody> element. If the RowGroup cell also contains a **CellScope** marker (or the `[HTMLParaStyles]/[StyleCellScope]` equivalent) that sets the `scope="rowgroup"` attribute, the RowGroup property works in concert with the `scope` attribute to apply the RowGroup header to all cells in its <tbody> section. The `scope` attribute is in effect only within the same <tbody> section as the RowGroup cell. See §27.2 [Using the scope method to identify table cells](#) on page 775.

If `RowGroupIDs=Yes`, each RowGroup cell is given an `id` consisting of the RowGroupHead name followed by a sequential number. This `id` is used as a `headers` attribute in all cells to the right of the RowGroup cell and all cells below that row (see [Figure 27-1](#) on page 780), until the next cell down that contains a RowGroup paragraph. See §27.3.3 [Grouping header cells for identification](#) on page 778.

28 Working with macros

You can use macros to insert any content into the output stream. Because the **Mif2Go** macro language is Turing-complete, the **Mif2Go** macro facility is powerful enough to let you insert anything in RTF output, and do almost anything to HTML or XML output. Topics include:

- §28.1 [Defining and invoking macros](#) on page 787
- §28.2 [Accessing Mif2Go macro libraries](#) on page 792
- §28.3 [Using macro variables](#) on page 795
- §28.4 [Using multiple-value list variables](#) on page 806
- §28.5 [Accessing settings with configuration macros](#) on page 809
- §28.6 [Using expressions in macros](#) on page 811
- §28.7 [Passing a parameter to a macro](#) on page 820
- §28.8 [Debugging macros](#) on page 820
- §28.9 [Deploying macros and macro variables](#) on page 820
- §28.10 [Using macros to fine-tune HTML or XML output](#) on page 828

See also:

- §33.2.3 [Overriding settings with macros](#) on page 921

28.1 Defining and invoking macros

In this section:

- §28.1.1 [Defining macros](#) on page 787
- §28.1.2 [Invoking a macro](#) on page 791
- §28.1.3 [Nesting macros](#) on page 791
- §28.1.4 [Using predefined macros](#) on page 792

28.1.1 Defining macros

To define a macro, create a configuration-file section with the name of the macro as the section name. This section can go in your project configuration file, or in a macro library file; see §28.1.1.2 [Understanding where you can define named macros](#) on page 788.

For example, to define macro `$OurLogo` for HTML output:

```
[OurLogo]  
<hr /><br /><br /><hr /><br />
```

The content of the macro begins on the next line after the section name, and ends at the start of the next section, or at the end of the configuration file. The name must consist only of letters and digits. Do not include punctuation or spaces in a macro name.

In this section:

- §28.1.1.1 [Understanding what a macro definition can include](#) on page 788
- §28.1.1.2 [Understanding where you can define named macros](#) on page 788
- §28.1.1.3 [Escaping special characters in macro definitions](#) on page 789
- §28.1.1.4 [Managing line breaks in macro definitions](#) on page 789
- §28.1.1.5 [Including comments in macro definitions](#) on page 789
- §28.1.1.6 [Including cross references in macro definitions](#) on page 790

§28.1.1.7 [Converting “smart quotes” in macro definitions](#) on page 790

§28.1.1.8 [Obtaining RTF code for macro definitions](#) on page 790

28.1.1.1 Understanding what a macro definition can include

Macros can include more than simple code:

- For HTML output a macro can contain HTML code, JavaScript, or anything else printable that follows the rules of whatever language you are using.
- For RTF output, a macro can insert Field content or Windows system commands.
- For all types of output, a macro can specify Windows system commands, provided double any backslashes and enclose paths that contain spaces in double quotes; see §34.4.6 [Supplying system commands in a macro](#) on page 940.

A macro can be any length. You can define macros to use as “building blocks” for other macros. There is no limit to the number of macros you can define for a project.

Note: You do not have to define *every* string of code as a macro. Any place in the configuration file where you can use a macro, you can also use plain HTML or RTF code, *provided you include the entire code string on one line.*

Whether you use a formal named macro definition or an informal string of code, for HTML output **Mif2Go** always inserts an extra line break in the output immediately before the expanded macro. This is so you can readily identify macro-supplied code, for ease in correcting any errors in your macro settings. Browsers ignore the extra line break.

28.1.1.2 Understanding where you can define named macros

You can put **Mif2Go** macro definitions in any of the following places:

- **Best place:** in a macro library file; see §28.2 [Accessing Mif2Go macro libraries](#) on page 792.
- **If large or complex:** individually in separate macro files; see §28.2.3 [Storing a macro definition in a separate file](#) on page 793.
- **Otherwise:** toward the end of your project configuration file, before any [MacroVariables] section.

Order does not matter

The relative order in which macro definitions appear in a file is not important; what matters is the order in which they are invoked during conversion (see §28.1.2 [Invoking a macro](#) on page 791).

Do not end a file with a macro

Do not put a macro at the very end of a configuration file or library file. If you have no macro variables to define, and no [MacroVariables] section, end the file with a dummy section; for example:

[End]

No macros in templates

Do not include macro definitions in a configuration template (see §30.6.2 [Deciding what to include in a general configuration template](#) on page 862).

Put complex macros in a separate file

If you create lengthy macros (for example, with a lot of conditional expressions), and you indent the code for readability, put the macros in a library file separate from the configuration file; or put each macro in its own *macroname.txt* file. That way the indentation is preserved. When **Mif2Go** updates your project configuration file as a consequence of changes you make to Export options, Windows rewrites the file, and deletes all leading spaces in the settings.

Note: Do not put **Mif2Go** macro definitions on the HTML reference page in your FrameMaker document; **Mif2Go** does not look there.

28.1.1.3 Escaping special characters in macro definitions

Use a backslash in a macro to escape other characters, such as “\”, “<”, “>”, “””, “\$”, “;” and “ ” (space). For example, if you need to start a macro content line with “[” or “;” (left bracket or semicolon), preface the line with a backslash, to keep the line from being treated as a comment or section head:

```
[MyMacro]
\; This is not a configuration-file comment
; This is a configuration-file comment
\[NotTheNextSection]
[TheNextSection]
```

To specify a trailing space at the end of a macro, insert any of the following:

```
two spaces
\ (a backslash followed by a space)
\~ (a backslash followed by a tilde).
```

The \~ convention is especially helpful, because it allows you to show that a space is unequivocally intended.

Make sure to escape the backslash itself if your macro includes path names. For example:

```
[MyGraphicFileCopy]
cd <$$_currpath>\\wrap
copy "c:\\my graphics\\*.jpg"
copy "c:\\more graphics\\*.jpg"
```

To include a comment in macro definitions, see §28.1.1.5 [Including comments in macro definitions](#) on page 789.

28.1.1.4 Managing line breaks in macro definitions

A macro definition does not have to be all on one line; **Mif2Go** ignores line breaks when processing macros. However, any implicit line breaks in the definition are retained in output when a macro is expanded.

To remove an implicit line break so it does not appear in the output, end the line in question with a backslash “\”.

To remove *all* implicit line breaks from macros upon expansion:

```
[Macros]
; OmitMacroReturns = No (default)
; or Yes (omit macro linebreaks in output)
OmitMacroReturns=Yes
```

Be aware that omitting all line breaks means that the code generated from each expanded macro—even JavaScript code—ends up all on one line in the output. Few browsers can handle the very long lines that might result.

If you specify `OmitMacroReturns=Yes`, but still need line breaks in some macros to keep line lengths reasonable in output, you can insert a C-style line terminator “\n” in the definition, even in the middle of a line, wherever you want an explicit line break in the output.

28.1.1.5 Including comments in macro definitions

Any line in a macro definition that *starts* with a semicolon (;) is treated as a configuration-file comment, even lines that would otherwise execute system commands:

```
[SomeMacro]
; This entire line is a comment, and so is the next:
```

```

; jhjar <$$currpath>\help ugmif2go
; But the following line will be executed:
jhindex <$$currpath>\help html

```

Normally, a line that starts with a semicolon in a macro definition does not appear at all in the output. If you *do* want such a comment to appear in the output, as itself, escape the semicolon with a backslash:

```
\; my macro comment
```

When you do this, you get the backslash character in the output, which appears to be wrong based on the rule for escaping characters (see §28.1.1.3 [Escaping special characters in macro definitions](#) on page 789). However, using two backslashes “\\;” also results in “\;” in the output, which is correct.

There is a reason for this odd behavior. Macros can be nested, and it is desirable to avoid multiple escaping that depends on the nesting level. If the original backslash went away, and the macro was nested inside another macro, the comment would disappear on the next evaluation, unless you used “\\;”; and if the macro was nested two deep you would need “\\\\;”, which starts to become user unfriendly. Keeping the single backslash avoids all that, but it can cause astonishment.

What you do *not* get in the output is an HTML comment:

```
<!-- my macro comment -->
```

If that is what you want, put the comment in your macro using HTML comment syntax, exactly as you want it to appear in HTML output.

28.1.1.6 Including cross references in macro definitions

When a macro definition includes or is part of a forward cross reference to a FrameMaker file that **Mif2Go** will not yet have processed at the time the macro is expanded, **Mif2Go** cannot resolve the reference for HTML or XML output, because at that point the destination file does not exist. If your project employs macros of this type, you might have to run a conversion twice to resolve all cross references. See §C.5.2 [Resolving forward references with a second pass](#) on page 1028.

28.1.1.7 Converting “smart quotes” in macro definitions

Mif2Go can convert FrameMaker “smart quotes” (curly quotes), baseline quotes, and guillemets that are used within macros (typically in attribute values), into the straight quotes preferred by HTML browsers and by language interpreters such as those used for processing JavaScript code:

```

[Macros]
; FixMacroQuotes = No (default) or Yes (change curly quotes)
FixMacroQuotes=Yes

```

28.1.1.8 Obtaining RTF code for macro definitions

RTF coding is arcane, especially for tables. Unless you are an RTF expert, your best bet might be to copy existing RTF code. Here are some ways to obtain RTF code for your macros:

- [Get code examples from Word](#)
- [Get code examples from Mif2Go](#)
- [Generate RTF code with Mif2Go.](#)

Get code
examples from
Word

You can pretty-print RTF output from Word to mine for code (if you open a Word RTF file directly in a text editor, you see only unbroken lines of unreadable code):

1. In Word, create an example of the output you want.
2. Save as RTF from Word.
3. At a Windows command prompt, run pretty-printer program `pprtf.exe` on the saved RTF. The `pprtf.exe` program is included in your **Mif2Go** distribution directory.

The RTF pretty-printer, `pprtf.exe`, takes either one or two arguments:

- the name of the RTF file, with extension
- optionally, a different name for the output file, with extension

and creates a new file:

```
pprtf ExampleFile.rtf NewFile.txt
```

If you omit the second argument, the output is a file of the same name as the RTF file, but with extension `.txt`.

Get code
examples from
Mif2Go

Another way to obtain RTF code is to create an example in FrameMaker, run **Mif2Go**, and then copy/paste the resulting RTF code into your `m2rtf.ini` configuration file or into a macro library file. **Mif2Go** produces RTF output that is even more readable than the output from `pprtf.exe`.

Generate RTF
code with **Mif2Go**

For paragraphs, you can use `CodeStore` to generate RTF code; see §28.3.7.2 [Inserting code with the `CodeStore` property](#) on page 804.

28.1.2 Invoking a macro

To invoke a macro, insert its name, enclosed in a `<$ >` tag:

```
<$Macroname>
```

The dollar sign at the start of the tag is not valid in HTML, so it will not interfere with any real HTML (or XML) code. A space after the dollar sign is optional. When **Mif2Go** sees a macro name, it replaces the tag with the macro content.

You can invoke a macro:

- as all or part of the *value* in certain *key=value* configuration settings; see §28.9.1 [Understanding where to use macros and macro variables](#) on page 821.
- in a FrameMaker **HTML Macro** marker; see §28.9.7 [Using HTML Macro markers to invoke macros](#) on page 828.

Wherever you can invoke a macro, you can also supply plain HTML. You do not have to name and define strings of HTML code that you expect to include in only one place.

Invoking an
undefined macro

Mif2Go ignores the invocation of any macro for which no definition can be found, unless you specify a special debugging option; see §28.8 [Debugging macros](#) on page 820. You can take advantage of this behavior to set up a series of alternatives, then selectively enable only the ones you want for a given conversion project by renaming (or moving) macro library files. See §28.2.4 [Including macro definitions in your own macro library](#) on page 794.

28.1.3 Nesting macros

Within one macro you can invoke another macro, and that macro can invoke another, and so on; you can nest macro invocations to any level. When a macro calls another macro, **Mif2Go** notes the “nesting level” and compares it with the limit you set:

```
[Macros]
; MacroNestMax = maximum depth of macro calls in one statement
; used to prevent runaways when macros call each other in circles
MacroNestMax=128
```

So if you define a macro as:

```
[Again]
<P>Play it again, Sam.</P><$Again>
```

you would get at most 128 lines, then **Mif2Go** would continue. You cannot crash it by making it loop.

28.1.4 Using predefined macros

Mif2Go provides several predefined macros for HTML, listed in [Table 28-1](#).

*Predefined macro
names are
reserved*

Avoid giving any of your own macros a name that starts with an underscore; the **Mif2Go** definition takes precedence. The “\$” says “this is a **Mif2Go** construct”; the “_” says “the name is reserved, not one of yours”.

Note: For backward compatibility **Mif2Go** recognizes a predefined macro name *without* the underscore (such as `<$trail>`), but only if you have not defined your own macro with the same name. Your macro definition takes precedence if there is no underscore.

Table 28-1 Predefined macros for HTML output

Macro	Description	Ref.
<code><\$localtoc></code>	List of links to subordinate files (a local TOC)	20.3.3
<code><\$lastlocaltoc></code>	Copy of last local TOC generated	20.3.3
<code><\$madewith></code>	“Made with Mif2Go ” label or button	13.15
<code><\$next></code>	Link to, and title of, following file	20.4
<code><\$prev></code>	Link to, and title of, preceding file	20.4
<code><\$seqnext></code>	Link to, and title of, following file in the book	20.4
<code><\$seqprev></code>	Link to, and title of, preceding file in the book	20.4
<code><\$TopicStartCode></code>	Macros from marker type TopicStartCode	29.2.1
<code><\$trail></code>	Inserts a “breadcrumb trail” of links	20.2

You cannot use predefined macros in system commands. See §34.4.5 [Supplying system commands in a .bat file](#) on page 940.

28.2 Accessing Mif2Go macro libraries

In this section:

§28.2.1 [Understanding Mif2Go-supplied macro libraries](#) on page 792

§28.2.2 [Modifying Mif2Go-supplied macro definitions](#) on page 793

§28.2.3 [Storing a macro definition in a separate file](#) on page 793

§28.2.4 [Including macro definitions in your own macro library](#) on page 794

28.2.1 Understanding Mif2Go-supplied macro libraries

Your **Mif2Go** distribution includes several macro library files in the form of macro configuration templates, listed in [Table 30-7](#). These macro libraries are located in directory %OMSYSHOME%\m2g\macros. The templates are chained together by references.

To access a macro library (for example):

```
[Templates]
; Macros = path to macro library file
Macros = %OMSYSHOME%\m2g\m2htm_macro.ini
```

Mif2Go checks the referenced chain of macro libraries whenever a macro you invoke is not defined in your project configuration file.

A macro library file can include another [Templates]Macros setting, to make a chain of macro libraries to be searched; the chain can be any length. However, all files in the chain must have distinct names; the chain stops if **Mif2Go** finds a repeat.

Your **Mif2Go** project configuration file should reference m2htm_macros.ini or m2rtf_macros.ini, either directly or indirectly through your own macro library file.

See also:

§30.2 [Referencing configuration files and templates](#) on page 851

§28.2.4.3 [Creating a chain of macro libraries](#) on page 795

28.2.2 Modifying Mif2Go-supplied macro definitions

You can modify the macro definitions included in the macro libraries supplied with your **Mif2Go** distribution, located in %OMSYSHOME%\m2g\macros. However, if you change anything in those files, whenever you update **Mif2Go** you will need to run a file comparison program to see if anything has been added or changed by Omni Systems developers; see §1.3.8 [Obtain a file comparison tool \(optional\)](#) on page 60.

An alternative is to create your own macro library file and copy into it any macros you want to alter; see §28.2.4 [Including macro definitions in your own macro library](#) on page 794.

One sample macro is a proposed definition for a spacer for indenting graphics and tables. A macro variable is suggested for use with this macro (see §28.3 [Using macro variables](#) on page 795):

```
[Spacer]
" alt="[spacer]">

[MacroVariables]
spacerwidth=80
```

You can copy these definitions into your own macro library file, and modify them as you wish.

28.2.3 Storing a macro definition in a separate file

You might want to use individual files for very large macros; or a separate file for a macro that you want to include in different configurations, much like a text inset.

A macro file is a text file that contains a nameless macro, with content that comprises the definition of the macro. A macro file can have any name and any extension; however, it makes sense to give the file a base name that is the name you would have given the same macro if included in a macro library file.

To invoke a macro in a macro file, specify a path to the macro file inside a \$< . . . > wrapper. The path must include at least one path separator (forward slash or backslash); this is what distinguishes a file macro invocation from a local or library macro invocation. A relative path is relative to the project directory. For example:

```
[ParaStyleCodeReplace]
ParaFmt = <$. /mymacro.ini>
```

would cause **Mif2Go** to replace each instance of *ParaFmt* in the output with the content of *mymacro.txt*, located in the project directory.

A macro in a macro file can invoke other macros, including predefined macros, macros in other macro files, macros in library files, and macros in configuration files. Macros in macro files do not participate in the rules of precedence for chained macro libraries.

See also:

§28.2.4 [Including macro definitions in your own macro library](#) on page 794

28.2.4 Including macro definitions in your own macro library

You can construct a library of macros to use from anywhere in your project, or even across multiple projects, by storing macro definitions in a configuration file of their own: a macro library file. Macros in the library are defined the same way as in your project configuration file, each macro in its own section. If a macro definition is not present in your project configuration file, **Mif2Go** looks for the definition in a macro library file.

In this section:

§28.2.4.1 [Creating a macro library](#) on page 794

§28.2.4.2 [Creating a file-specific macro library](#) on page 795

§28.2.4.3 [Creating a chain of macro libraries](#) on page 795

28.2.4.1 Creating a macro library

To create your own macro library:

1. Create a new text file for your macro library. Give the file extension *.ini*, and either place it in your project directory or specify an absolute path to its location. It is a good idea to use the same location for macro library files for all your **Mif2Go** projects.
2. Add macro definitions to the file, each in its own section, as described in §28.1.1 [Defining macros](#) on page 787.
3. Put a non-macro dummy section at the end of the file; for example:
[End]

Otherwise, the last macro in the library might cause an extra character to be included in output.

4. Make your library file reference *m2rtf_macros.ini* or *m2htm_macros.ini*, and make your project configuration file reference your library file. For example, suppose you create a text file called *MyMacros.ini*, and place it in *D:\MacroLibs*.

In *MyMacros.ini*:

```
[Templates]
Macros = %OMSYSHOME%\m2g\m2htm_macro.ini
```

In your project configuration file:

```
[Templates]
Macros = D:\MacroLibs\MyMacros.ini
```

If you omit a path, **Mif2Go** looks for *MyMacros.ini* in your project directory.

Because *MyMacros.ini* is closer to your project configuration file in the chain of macro libraries, your macro definitions take precedence over any definitions of the same macros further away from your project configuration file in the chain.

Default macro library If you do not specify a value for `Macros`, and you invoke a macro that is not defined in your project configuration file, **Mif2Go** looks in `%OMSYSHOME%\m2g\macros` for a file named `m2htm_macros.ini` or `m2rtf_macros.ini`.

28.2.4.2 Creating a file-specific macro library

If you store macro definitions in a file named the same as the FrameMaker file you are converting, but with extension `.ini` instead of `.fm`, **Mif2Go** uses the macros in that file in place of any with the same macro names in your project configuration file. This lets you plug in file-specific code and data.

When you create a file-specific macro library, put a non-macro dummy section at the end of the file; for example:

[End]

Otherwise, the last macro in the library might cause an extra character to be inserted in the output.

28.2.4.3 Creating a chain of macro libraries

A macro library file can include a setting for `[Templates]Macros`, so the chain of libraries for **Mif2Go** to search for macro definitions can be any length. However, all files in the chain must have distinct names; the chain stops if **Mif2Go** finds a repeated macro library name.

Precedence of macro definitions

In a chain of macro libraries, if the same macro appears in more than one library file but has a different definition in each file:

- A definition in a library closer in the chain to the project configuration file overrides a definition in any library farther away in the chain.
- A definition in the project configuration file overrides the final library value.
- A definition in an individual chapter configuration file (see §33.1 [Using a different configuration for selected files](#) on page 919) overrides a definition in the project configuration file, *for that chapter only*.

Mif2Go builds a set of macros for each FrameMaker file in your project by starting with the most specific macro definitions: those in the `chapter.ini` configuration file, if there is one. Next come macro definitions in your project configuration file.

Next, if `chapter.ini` includes a value for `[Templates]Macros`, definitions in the referenced macro library (and any additional libraries chained to it) are applied. If `chapter.ini` does not reference a macro library, next come definitions in any macro library referenced by the project configuration file; then on up the chain from that library.

In other words, a chain of macro libraries is applied to `chapter.fm` either from `chapter.ini` (preferentially) or from the project configuration file, but not from both. In either case, definitions from a chain of macro libraries are applied after macro definitions from the project configuration file, which are applied after definitions from the chapter configuration file. For the same macro with different definitions in different configuration files or macro libraries, the definition in the most specific file takes precedence.

28.3 Using macro variables

In this section:

§28.3.1 [Creating and invoking macro variables](#) on page 796

§28.3.2 [Assigning values to macro variables](#) on page 797

§28.3.3 [Incrementing and decrementing macro variables](#) on page 799

§28.3.4 [Using predefined macro variables](#) on page 800

§28.3.5 [Treating FrameMaker user variables as macro variables](#) on page 801

§28.3.6 [Using some FrameMaker system variables as macro variables](#) on page 802

§28.3.7 [Creating macro variables from paragraph content](#) on page 802

28.3.1 Creating and invoking macro variables

In this section:

§28.3.1.1 [Naming macro variables](#) on page 796

§28.3.1.2 [Creating a macro variable](#) on page 796

§28.3.1.3 [Invoking a macro variable](#) on page 796

28.3.1.1 Naming macro variables

A **Mif2Go** macro variable name looks like a **Mif2Go** macro name, except that a macro variable name starts with *two* dollar signs instead of one: `$$varname`. The rest of the name must consist only of letters and digits. Do not include punctuation or spaces in a macro variable name.

*Reserved naming
for predefined
macro variables*

Some macro variable names are predefined by **Mif2Go**, and cannot be used for other purposes; see §28.3.4 [Using predefined macro variables](#) on page 800. The name of a *predefined* **Mif2Go** macro variable starts with two dollar signs followed by an underscore: `$$_varname`. Avoid giving a name that starts with an underscore to any of your own macro variables; the **Mif2Go** definition takes precedence. The “`$$`” says “this is a **Mif2Go** macro variable”; the “`_`” says “the name is reserved, not one of yours”.

Note: For backward compatibility **Mif2Go** recognizes a predefined variable name *without* the underscore (such as `$$basefile`), but only if you have not defined your own variable with the same name. Your variable definition takes precedence if there is no underscore.

28.3.1.2 Creating a macro variable

You create a **Mif2Go** macro variable when you do any of the following:

- Use the variable as the first term in a macro assignment or increment/decrement statement; see §28.3.2 [Assigning values to macro variables](#) on page 797.
- List the name of the variable in [MacroVariables] (for use in macros); see §28.3.2 [Assigning values to macro variables](#) on page 797.
- List the name of the variable in one of the following configuration-file sections:
 - [MacroVariables] (for use in macros); see §28.3.2 [Assigning values to macro variables](#) on page 797
 - [UserVars] (for use in system commands); see §34.5.1 [Assigning an initial value to a user variable](#) on page 941
- Define a FrameMaker user variable in your document; see §28.3.5 [Treating FrameMaker user variables as macro variables](#) on page 801
- Assign a TextStore or CodeStore property to a paragraph format; see §28.3.7 [Creating macro variables from paragraph content](#) on page 802.

28.3.1.3 Invoking a macro variable

You invoke a macro variable like this:

```
<$$varname>
```

Or like this:

```
<$$varname as display-format>
```

where *display-format* is a C-language style `printf()` format. See §28.6.3 [Displaying expression results in output](#) on page 813.

You do not need the enclosing angle brackets when you use a macro variable inside a macro; for example, in an assignment such as `<$$myvar = ($$othervar + 2)>`.

An example

Suppose you want to use a macro that includes the following:

- an image, but with a different `src` attribute each time
- a heading, but with different text each time.

Rather than have two almost identical macros, you can use a macro variable for the `src` attribute and another for the heading, then set their values appropriately for each use.

You could define the macro like this:

```
[TopStory]
<h2><$$Head></h2>
```

Call it like this one day:

```
<$$Pic=lead000201.jpg><$$Head=No Survivors in Crash><TopStory>
```

and like this the next day:

```
<$$Pic=lead000202.jpg><$$Head=MS Embraces Linux><TopStory>
```

28.3.2 Assigning values to macro variables

You can initialize the value of a macro variable in your configuration file, and you can assign a value to a macro variable in the body of a macro definition:

[Assign a starting value](#)

[Assign a value in a macro](#)

[Assign a character literal.](#)

Assign a starting value

Assign starting values to macro variables in configuration section `[MacroVariables]`. Omit the leading `$$` when you specify the name. For example:

```
[MacroVariables]
; varname = value to use as literal replacement, can be in Macro Ini
; can also be set in any macro with <$$name=value>, settings persist
; until the end of the file, but are not stored in the .ini file.
HdgCount = 000
HdgColor = blue
```

You can assign only literal values; you cannot assign a value that specifies a macro or another macro variable.

Place section `[MacroVariables]` in one (or more) of the following files, after any macro definitions:

- your project configuration file
- a configuration template
- a separate macro file or macro library file.

Assign a value in a macro

Use any of the following forms to assign values to variables inside **Mif2Go** macros:

```
<$$varname = $$othername>

<$$varname = (expr)> (See §28.6 Using expressions in macros on page 811)

<$$varname = "quoted string even with \"double quotes\" in it">

<$$varname = 'quoted string using "single" quotes'>

<$$varname = string with no quotes>
```

`<$$varname = 'x'>` (Character literal)

Assign a
character literal

The value of a character literal assigned to a macro variable is the ASCII value of the character. A character literal can be a character enclosed in single quotes, or any of the special cases listed in [Table 28-2](#).

Table 28-2 Character literals for macro variables

Character literal	Name	Decimal ASCII value
'\r'	return	13
'\n'	newline	10
' '	empty	0
'\''	single quote	39
'\\'	backslash	92

Characters other than ' and \ that are preceded by a backslash are themselves. However, ' and \, *without* a backslash, are *not* themselves:

' ' would be an empty string followed by an out-of-place ', thus 0 (zero)

'\' is invalid, and would probably become a string with a single quote, equivalent to "\"

When a string between single quotes contains more than two characters (or more than one when the first character is *not* a backslash), you do not have to escape double quotes *within* the string, a common JavaScript and HTML technique.

Display an
assignment

Assigning a value to a macro variable does not cause the value to appear in output. To display the value of an assignment, use **as** and a `printf()` format. For example, if the value of `<$$myvar>` is 0 (zero), the following expression displays the value 0001:

```
<$$myvar = ($$myvar + 1) as %0.4d>
```

See [§28.6.3 Displaying expression results in output](#) on page 813.

Assign a value
indirectly

You can assign a value to a variable indirectly:

```
<$$myvar = "$$other">
<*$myvar = 10>
```

This sequence results in assigning the value 10 to `$$other` rather than to `$$myvar`. See [§28.6.7 Using indirection in expressions](#) on page 819.

Nest macro
variables

You can nest macro variables:

```
[Macros]
; MacroVarNesting = Yes (default, vars contain <>)
; or No (first > ends var)
MacroVarNesting=Yes
```

This setting is provided solely to support old syntax in assignments. You used to use:

```
<$$myvar=<$$othervar>>
```

to get what is now simply:

```
<$$myvar = $$othervar>
```

You need `MacroVarNesting=Yes` only if your macro variable assignments use the old syntax; the new syntax is always valid. Either way, you get the *contents* of the referenced right-hand variable, rather than its name.

Note: Macro variables cannot contain macros.

See also:

§28.4.2 [Assigning a value to a list-variable item](#) on page 806

§33.2.4 [Assigning values to configuration variables](#) on page 922

28.3.3 Incrementing and decrementing macro variables

You can increment the value of a macro variable by 1 (one) like this:

```
<$$myvar++>      (or just <$$myvar+>)
```

or decrement the value by 1 like this:

```
<$$myvar-->      (or just <$$myvar->)
```

For example, to count *Body* paragraphs in a FrameMaker file for HTML output, incrementing the count before using it:

```
[HTMLParaStyles]
Body=CodeStart

[ParaStyleCodeStart]
Body=<!-- this is <$$bodynum++ as %0.3d> -->

[MacroVariables]
bodynum=bp000
```

These settings result in a comment like the following for each instance of a *Body* paragraph in the HTML output:

```
<!-- this is bp003 -->
```

*Reserve enough
digits*

You must include enough placeholder digits in the starting value (in this example, bp000) to accommodate the range of values you expect in the file. If you do not, the number will roll over to zero after it reaches its maximum value: in this example bp999 would increment to bp000. If the value has no digits at all at the end, the last letter is incremented instead; so a starting value of aaa increments to aab, aac, ..., aaz, aba, ..., zzz, aaa. Case is retained for the incremented (or decremented) letter.

To increment the value *after* use, move the incrementing code after the reference:

```
[ParaStyleCodeStart]
Body=<!-- this is <$$bodynum> --><$$bodynum++>
```

*Reset the starting
value*

Numbers restart for each FrameMaker file. If you require the numbers to be unique in your **Mif2Go** project, you must use an individual *FMfilename.ini* configuration file for each FrameMaker file in your document, and include in it a [MacroVariables] section with a starting value for that file.

*Increment by
assignment*

You can use an assignment (see §28.3.2 [Assigning values to macro variables](#) on page 797) as another form of increment, as in the following:

```
<$$myvar = ($$myvar + 1)>
```

This form does not require reserving the maximum number of digits first.

*Increment
hexadecimal
numbers*

Incrementing and decrementing using ++ or -- notation does not work with values stored as hexadecimal numbers; for those you *must* use an assignment to increment or decrement:

```
<$$myhex = ($$myhex + 1)>
```

*Display an
increment*

You can also display the value of an increment or decrement by adding **as** and a `printf()` format; for example:

```
<$$myvar++ as %d>
```

```
<$$myvar = ($$myvar + 1) as %0.4d>
```


See §28.6.3 [Displaying expression results in output](#) on page 813 for information about display formats.

*Increment
indirectly*

You can increment a variable indirectly:

```
<$$myvar = "$$other">
<*$$myvar++>
```

This sequence increments the value of `$$other` rather than the value of `$$myvar`. See §28.6.7 [Using indirection in expressions](#) on page 819.

28.3.4 Using predefined macro variables

Mif2Go provides a collection of predefined macro variables and user variables, listed in [Table 28-3](#). Every predefined macro variable name begins with “`$$_`”. Predefined macro variables are read-only; you cannot assign values to them, and you cannot increment or decrement them. However, you *can* do the following:

- Use predefined macro variables in expressions; see §28.6.1 [Understanding macro expressions](#) on page 811).
- Format output of a predefined macro variable; see §28.6.3 [Displaying expression results in output](#) on page 813).

Note: Only `<$$_basename>`, `<$$_currpath>`, and `<$$_prjpath>` can be used in system commands; other predefined macro variables do not work in system commands. See §34.4.6 [Supplying system commands in a macro](#) on page 940.

Table 28-3 Predefined macro variables

Macro variable	Where used	Description	Ref
<code>\$\$_basefile</code>	HTML split files	Base name only of parent file, without extension	18.6
<code>\$\$_basename</code>	System commands	Base file name (without path or extension) of current FrameMaker file or book	34.4.2
<code>\$\$_basetitle</code>	HTML split files	Original document title, unaffected by splits	18.6
<code>\$\$_chapnum</code>	Macros	FrameMaker system-variable building block	28.3.6
<code>\$\$_class</code>	Elements (HTML)	CSS class name of current paragraph	28.6.6
<code>\$\$_count</code>	Loop constructs	Current iteration count for <code><\$_repeat></code> loops	28.6.4.3
<code>\$\$_currbase</code>	Output files	File name of current file, without extension	18.6
<code>\$\$_currfile</code>	Output files	File name of current file, with extension,	18.6
<code>\$\$_currfilepath</code>	Output files	Path and name of current file, with extension	18.6
<code>\$\$_currpath</code>	System commands	Path, without trailing slash, to project directory	34.4.2
<code>\$\$_currtitle</code>	HTML split files	Current file title, unaffected by extracts	18.6
<code>\$\$_dcount</code>	Loop constructs	Down-count for <code><\$_repeat></code> loops	28.6.4.3
<code>\$\$_ditastart</code>	DITA XML production	Start tag of the current topic	15.4.3.6
<code>\$\$_element</code>	Elements (HTML)	Name of the current element	28.6.6
<code>\$\$_extrfile</code>	HTML extract files	File name of extracted file	18.7.3
<code>\$\$_extrgraph</code>	HTML extract files	File name of first extracted graphic	18.7.3
<code>\$\$_extrtitle</code>	HTML extract files	Title of extracted file	18.7.3
<code>\$\$_fileid</code>	Output files	FileID of FrameMaker file, from <code>mif2go.ini</code>	34.8.5
<code>\$\$_firstfile</code>	HTML split files	1 if first split part after original file, otherwise 0	18.6
<code>\$\$_graphbase</code>	HTML graphics	File name for <code></code> attribute, no extension	23.5.2
<code>\$\$_graphorighigh</code>	HTML graphics	Original height in pixels of the image	23.5.2
<code>\$\$_graphorigwide</code>	HTML graphics	Original width in pixels of the image	23.5.2
<code>\$\$_graphsrc</code>	HTML graphics	File name for <code></code> attribute, with extension	23.5.2

Table 28-3 Predefined macro variables (continued)

Macro variable	Where used	Description	Ref
<code>\$\$_lastfile</code>	HTML split files	1 if last part (regardless of splitting), or if unsplit; otherwise 0	18.6
<code>\$\$_linksrc</code>	HTML link attributes	<code>href</code> content of a link	19.2.4
<code>\$\$_loctocfile</code>	HTML split files	File name of subordinate file in local TOC entry	20.3
<code>\$\$_loctoctitle</code>	HTML split files	Title of subordinate file in local TOC entry	20.3
<code>\$\$_macroparam</code>	Macros	Value of parameter passed in parentheses	28.7
<code>\$\$_nextfile</code>	HTML split files	File name of split part that follows <code>\$\$_currfile</code>	18.6
<code>\$\$_nexttitle</code>	HTML split files	Title of <code>\$\$_nextfile</code> split part	18.6
<code>\$\$_objectid</code>	Marker reference	Object ID of current marker (HTML only)	29.8
<code>\$\$_paratag</code>	Formats	Name of current paragraph format; same as FrameMaker <code><\$paratag></code> building block	28.6.6
<code>\$\$_parauid</code>	HTML link anchors	“X” followed by FileID (if any) followed by ObjectID of current paragraph	19.3.4
<code>\$\$_prevfile</code>	HTML split files	File name of split part that precedes <code>\$\$_currfile</code>	18.6
<code>\$\$_prevtitle</code>	HTML split files	Title of <code>\$\$_prevfile</code> split part	18.6
<code>\$\$_prjname</code>	Macros	Base name of Mif2Go .prj file	
<code>\$\$_prjpath</code>	System commands	Path (without trailing slash) to the directory where the .prj file resides	34.4
<code>\$\$_sectionnum</code>	Macros	FrameMaker 9+ system-variable building block	28.3.6
<code>\$\$_seqcurrtitle</code>	HTML navigation links	Title of current file	20.4
<code>\$\$_seqnextfile</code>	HTML navigation links	Name of following file, with extension .htm	20.4
<code>\$\$_seqnexttitle</code>	HTML navigation links	Title of following file	20.4
<code>\$\$_seqprevfile</code>	HTML navigation links	Name of preceding file, with extension .htm	20.4
<code>\$\$_seqprevtitle</code>	HTML navigation links	Title of preceding file	20.4
<code>\$\$_splitid</code>	HTML split files	Base name of split file, excluding FileID portion	34.8.5
<code>\$\$_splitnum</code>	HTML split files	Sequential number in file-name prefix or suffix	34.8.5
<code>\$\$_subsectionnum</code>	Macros	FrameMaker 9+ system-variable building block	28.3.6
<code>\$\$_tblcols</code>	Tables	Count of columns in current table	24.6.6
<code>\$\$_volnum</code>	Macros	FrameMaker system-variable building block	28.3.6
<code>\$\$_tblrows</code>	Tables	Count of rows in current table	24.6.6
<code>\$\$_wcount</code>	Loop constructs	Iteration count for <code><\$_while></code> loops	28.6.4.3
<code>\$\$_xrefid</code>	HTML link anchors	“R” followed by FileID (if any) followed by ObjectID of first cross-reference marker in current paragraph	19.3.4

28.3.5 Treating FrameMaker user variables as macro variables

You can employ a FrameMaker variable as a **Mif2Go** macro variable in configuration settings, provided the name of the variable is compatible with **Mif2Go** macro variable naming rules; that is, the name starts with a letter and contains no spaces or punctuation except underscores. If necessary, you can modify the name of a FrameMaker variable to fit the rules, as follows:

1. Remove all punctuation from the FrameMaker variable name except underscores.
2. Replace spaces in the name with underscores.
3. If the first character is a digit, prefix the name with “x”.

For example, if a FrameMaker user variable in your document is named *2nd-Best Choice*, you must specify it in configuration settings as `x2ndBest_Choice` for **Mif2Go** to

recognize it as referring to that particular FrameMaker variable; and you would use the variable in **Mif2Go** macros as `<$$x2ndBest_Choice>`.

When **Mif2Go** encounters in your configuration file a macro variable name that occurs in *none* of the following places:

- as the first term in an assignment
- in [MacroVariables]
- in [UserVars] (see §34.5.1 [Assigning an initial value to a user variable](#) on page 941)

Mif2Go looks in your document for a FrameMaker variable that fits that name, possibly modified according to rules 1 through 3 above. If **Mif2Go** finds such a variable, **Mif2Go** creates a macro variable of the same name (prefixed with \$\$) and with the same value, just as though you listed the variable and a starting value for it in [MacroVariables] or in [UserVars]. However, any formatting in the FrameMaker definition is lost.

For example, suppose you define a macro such as the following:

```
[DocTitle]
<p>Programmer's Guide, Version <$$Vnum></p>
```

If your FrameMaker document contains a user variable named *Vnum*, and you do not explicitly create a macro variable named `$$Vnum` in the configuration file, **Mif2Go** uses the value of FrameMaker user variable *Vnum* for **Mif2Go** macro variable `$$Vnum`.

See also:

- §28.3.6 [Using some FrameMaker system variables as macro variables](#) on page 802
- §5.4 [Applying FrameMaker conditions and variables](#) on page 122

28.3.6 Using some FrameMaker system variables as macro variables

Mif2Go provides predefined macro variables for the following FrameMaker system-variable building blocks:

<code><\$\$_volnum></code>	<i>Volume Number</i>
<code><\$\$_chapnum></code>	<i>Chapter Number</i>
<code><\$\$_sectionnum></code>	<i>Section Number</i> (FrameMaker version 9+ only)
<code><\$\$_subsectionnum></code>	<i>Subsection Number</i> (FrameMaker version 9+ only)

These four are the only FrameMaker system variables that can be used as **Mif2Go** macro variables. Because references to these FrameMaker system-variable building blocks follow the same naming rules as FrameMaker user variables, in a **Mif2Go** macro you could refer to the value of (for example) the FrameMaker *Chapter Number* variable in either form: `<$$Chapter_Number>` or `<$$_chapnum>`.

See also:

- §28.3.5 [Treating FrameMaker user variables as macro variables](#) on page 801

28.3.7 Creating macro variables from paragraph content

Two [*Styles] properties, *TextStore* and *CodeStore*, allow you to assign text or code to a paragraph format, and have the content of any paragraph in that format stored in a macro variable for later insertion in the output.

In this section:

- §28.3.7.1 [Capturing paragraph content with the TextStore property](#) on page 803
- §28.3.7.2 [Inserting code with the CodeStore property](#) on page 804
- §28.3.7.3 [Understanding why TextStore and CodeStore work differently](#) on page 805

28.3.7.1 Capturing paragraph content with the TextStore property

To store the text content of a FrameMaker paragraph in a macro variable, assign the `TextStore` property to the paragraph format:

```
[HTMLParaStyles]
; TextStore stores the paragraph content as plain text in the
;   macro variable named in [StyleTextStore].
ParaFmt = TextStore
```

Explicitly assigning the `TextStore` property to a paragraph format is optional when you assign a macro variable to that format in the following section:

```
[StyleTextStore]
; doc para format = name of macro variable in which to store para text
;   if omitted, default is a macro variable of the para format name
ParaFmt = Varname
```

<i>Format name is default variable name</i>	If you assign the <code>TextStore</code> property to a paragraph format, but you do not supply a macro variable name in section <code>[StyleTextStore]</code> , Mif2Go uses the paragraph format name itself as the macro variable name.
<i>Plain text</i>	<code>TextStore</code> macro variables contain just plain text; no HTML tags, RTF formatting code, macros, frames, or tables. Although the original paragraph content is left in place, you can suppress its appearance in output by also assigning the <code>Delete</code> property to the paragraph format.
<i>Only last instance counts</i>	If more than one instance of a <code>TextStore</code> paragraph format appears in a portion of your document destined for a given split or extract file, the <code>TextStore</code> macro variable retains the content of only the last instance, for that particular split or extracted file.
<i>Location can follow point of use</i>	For HTML, you can place a <code>TextStore</code> paragraph anywhere with respect to where you want the macro-variable content to be used, within the limits of material to be split or extracted into a single HTML output file; this is different from <code>CodeStore</code> paragraphs, which must <i>precede</i> the point of use (see §28.3.7.3 Understanding why TextStore and CodeStore work differently on page 805).
<i>Content is persistent</i>	The content of a <code>TextStore</code> macro variable persists unchanged in, and is available throughout, each HTML output file. If there is no instance of the paragraph format in the current split file, Mif2Go uses the content of the previous instance (or even a later instance) rather than come up empty-handed. Therefore, to prevent its use in a given split file, you must set the value to zero in that portion of the source document.
<i>Use for HTML navigation links</i>	Suppose you want your HTML output to include Prev and Next links between FrameMaker chapter files. You could create two special paragraph formats for this purpose; for example, <i>PrevChap</i> and <i>NextChap</i> . In each FrameMaker file you would include a single <i>PrevChap</i> paragraph containing the file name (base file name with extension <code>.htm</code> instead of <code>.fm</code>) of the preceding chapter, and a single <i>NextChap</i> paragraph containing the file name of the following chapter.

For example, suppose your book file contains the following files:

```
Title.fm
Preface.fm
ChapOne.fm
ChapTwo.fm
...
```

You might give `ChapOne.fm` a *PrevChap* paragraph with content `Preface.htm`, and a *NextChap* paragraph with content `ChapTwo.htm`. Most likely you would apply a condition to these paragraphs to prevent them from appearing in printed output.

In the configuration file you would assign the `TextStore` property to each of these paragraph formats, and include macros for the between-chapter **Prev** and **Next** links:

```
[HTMLParaStyles]
PrevChap=TextStore Delete
NextChap=TextStore Delete

[AtStart]
<p><a href="$$PrevChap">Prev</a></p>

[AtEnd]
<p><a href="$$NextChap">Next</a></p>
```

The `Delete` property ensures that *PrevChap* and *NextChap* paragraph content does not also appear in the output as regular text.

28.3.7.2 Inserting code with the `CodeStore` property

To store the content of a `FrameMaker` paragraph in a macro variable, assign the `CodeStore` property to the paragraph format:

```
[HTMLParaStyles] or [HelpStyles] or [WordStyles]
; CodeStore causes the paragraph content to be stored in the macro
; variable named in [StyleCodeStore]; the para must *precede*
; the point of use of the macro variable in the output document.
; The original para is removed; it can be put back by invoking the
; macro variable in a CodeAfter macro. Note that any CodeStart and
; CodeEnd macros are included in the macro variable content, but
; CodeBefore, CodeAfter, frames, and tables are not.
Parafmt=CodeStore
```

Explicitly assigning the `CodeStore` property to a paragraph format is optional when you assign a macro variable to that format in the following section:

```
[StyleCodeStore]
; doc para format = name of macro variable in which to store para text
Parafmt=Varname
```

Any `CodeStart` and `CodeEnd` macros are included in the macro variable content; however, `CodeBefore` macros, `CodeAfter` macros, frames, and tables are not included.

*Format name =
variable name*

If you assign the `CodeStore` property to a paragraph format, but you do not supply a macro variable name in `[StyleCodeStore]`, **Mif2Go** uses the paragraph format name itself as the macro variable name.

*Must precede
point of use*

A paragraph with the content you want to appear at a certain point in the output must be the last instance in your `FrameMaker` document that *precedes* the point where you want the content inserted. The macro variable holds the value of each instance of the paragraph format in turn until the point of insertion, whereupon **Mif2Go** inserts the latest value in the output. You can give the macro variable a starting value by defining its name in section `[MacroVariables]`; see §28.3.2 [Assigning values to macro variables](#) on page 797.

*Content is
ignored*

When you assign the `CodeStore` property to a paragraph format, **Mif2Go** removes all instances of text in that format from the output. You can restore the text by invoking the macro variable in a `CodeAfter` macro.

Observe the following caveats:

- Do not assign property `Delete` to the `CodeStore` paragraph format; if you do, the paragraph content will *not* be stored in the macro variable.
- Avoid assigning `CodeStore` to a paragraph format that has any of the HTML `ListN` properties; the `` coding will be misplaced.

*Insert HTML
navigation links*

Suppose you have a paragraph format named *Nextsect* that you use for cross references to other `FrameMaker` files in your book. And suppose you want to save the text of any

Nextsect paragraph in macro variable <\$\$Footnext>, so you can make the link appear in a footer in the HTML output. You would specify these settings:

```
[HTMLParaStyles]
Nextsect=CodeStore

[StyleCodeStore]
Nextsect=Footnext
```

The content of each successive paragraph in format *Nextsect* would be stored in macro variable <\$\$Footnext>, replacing the previous value for each instance of *Nextsect*. Wherever you insert macro variable <\$\$Footnext>, its current content, taken from the latest instance of *Nextsect*, appears in the output.

Generate RTF code

You can use *CodeStore* to generate RTF code for use in later macros, so you do not have to know arcane RTF syntax. For example, suppose you want a copyright notice at the bottom of every WinHelp topic. Put the notice in a paragraph at the start of each chapter file in your FrameMaker document, using a special paragraph format (for example, *Copyr*). Make the *Copyr* paragraphs conditional for “Help only” to keep the notices out of print versions of your document. Then specify the following configuration settings:

```
[HelpStyles]
Copyr=CodeStore

[Inserts]
TopicEnd=<$$Copyr>
```

This is much easier than trying to compose RTF code yourself to insert via macro:

```
[Copyr]
\pard \s12 \f3 \fs20 \b Copyright \'a9 2012 Softworks Inc. \par
```

Besides, code such as \s12 and \f3 could change from one run to the next.

28.3.7.3 Understanding why TextStore and CodeStore work differently

Mif2Go DCL “write” filters, such as *dwhtml.dll*, operate in two phases:

- Scan phase:* The results of converting MIF to DCL (see §1.5 [How Mif2Go works](#) on page 62) are stored in linked lists in memory.
- Write phase:* The linked lists in memory are traversed, additional information is collected, and output files are written.

<i>TextStore</i>	<i>TextStore</i> information is set during scan phase, and is available during write phase.
<i>CodeStore</i>	<i>CodeStore</i> information is set during write phase, and is available only after the point in the FrameMaker file where the <i>CodeStore</i> paragraph occurs.
<i>Scan phase</i>	During scan phase, information needed to produce the final output is incomplete. For example, links to other files (including links among split files) are not resolved until the <i>end</i> of the scan phase. <i>TextStore</i> processing is able to save the text content of a paragraph during scan phase, because the text content is known at that time. The <i>TextStore</i> property excludes items that are not known, such as links. This is the same mechanism used to generate <i>Title</i> content.
<i>Write phase</i>	During write phase, Mif2Go makes numerous cross-list accesses to pick up bits of information needed to build the final output. Links are completed, macros are executed, tables are constructed, graphics names are determined, and coded text is generated. <i>CodeStore</i> processing saves the coded text from a paragraph assigned the <i>CodeStore</i> property; however, at that point previous paragraphs have already been written to final output, and cannot be altered.

28.4 Using multiple-value list variables

In addition to single-value macro variables (see §28.3 [Using macro variables](#) on page 795), you can use multiple-value list variables. A *list variable* is a macro variable that contains an ordered, indexable collection of items, each of the form *index=value*, much like an array in the C programming language. A list variable can hold up to 64K items.

In this section:

§28.4.1 [Understanding list-variable syntax](#) on page 806

§28.4.2 [Assigning a value to a list-variable item](#) on page 806

§28.4.3 [Initializing list variables](#) on page 807

§28.4.4 [Using macros to process lists](#) on page 807

§28.4.5 [Using pointers to process lists](#) on page 808

§28.4.6 [Using a list instead of a conditional expression](#) on page 809

28.4.1 Understanding list-variable syntax

To create a list variable, all you have to do is use a **Mif2Go** macro variable name with an index value in brackets, similar to C-language array notation:

```
$$listname[index]
```

For example:

```
$$mylist[$$_count]      a variable as the index
$$mylist[2]             a constant as the index
$$mylist[( $$myindex + 1)] an expression as the index
```

*List indexes can
be nested*

The index is a string, not just a number, so it can be anything, even another nested list reference:

```
<$$mylist[$$another[one]]>
```

You can access the number of items in the list with `<$$mylist[]>`.

28.4.2 Assigning a value to a list-variable item

To specify the value of an item in a list, use an assignment that includes the index position of the item in brackets (see §28.3.2 [Assigning values to macro variables](#) on page 797):

```
<$$listname[index] = somevalue>
```

How the value is assigned depends on whether you include or omit a default value:

[Set a default value](#)

[Omit a default value.](#)

*Set a default
value*

If you provide a value for list item 0 (zero), that value is used for any item in the list for which you have not specified another value. For example, suppose you specify:

```
<$$mylist[0] = Error!>
```

Then if you use `<$$mylist[15]>`, without ever having assigned a value to the 15th item in the list, that value becomes “Error!”.

*Omit a default
value*

If you do *not* provide a value for `<$$mylist[0]>`, and you use `<$$mylist[15]>`, the value of `<$$mylist[15]>` is 0 (zero).

28.4.3 Initializing list variables

To preset values for list-variable items, create a configuration-file section for the list variable, and use the section to populate the list with indexes and corresponding values. For example:

```
[MyList]
1 = First
abc = Alpha start
xyz = Alpha end
0 = Default
mno = middle
```

With these settings, the initial value of `<$$MyList[1]>` would be `First` and the initial value of `<$$MyList[mno]>` would be `middle`. The initial value of `<$$MyList[ghi]>` would be `Default`, because no value has been provided for an index named `ghi` (see §28.4.2 [Assigning a value to a list-variable item](#) on page 806).

Suppose you set `<$$MyList[1]=Second>` in a macro. If this macro or another macro subsequently refers to `<$$MyList[1]>` while **Mif2Go** is processing the same FrameMaker file, the value is `Second`, not `First`. But the value in the configuration file does not change, so when **Mif2Go** processes the next FrameMaker file, the initial value of `<$$MyList[1]>` is `First` again. That is, the new value `Second` is in effect for the rest of the current FrameMaker file, but is not stored for use in the next.

28.4.4 Using macros to process lists

Suppose you want to generate a different set of sidebar navigation links for each major section of a Web site, where each section is in a single FrameMaker file that **Mif2Go** splits into named pages (see §18 [Splitting and extracting files](#) on page 585). For each page, the sidebar item that names that page should *not* be a link, because a live link to the current page confuses people.

You need a slightly different list of sidebar items on every HTML page, to avoid a same-page link. But the logic is always the same, as are the names of files and the titles to be displayed. What is needed is a macro that takes into account which item should not be linked.

Your configuration file could include a pair of lists, one with file names and the other with matching sidebar titles, like this:

```
[FM_File]
1 = homepage
2 = descript
3 = operate
4 = testimonial
5 = demo
6 = order

[SideTitle]
1 = Widgets
2 = What a Widget Does
3 = How to Use a Widget
4 = What Users Say About Widgets
5 = Get a Demo Widget
6 = Order Widgets On Line
```

You could process the two lists with this macro:

```
[Sidebar]
<$$val=2><$$maxval=6>\
<$_while ($$val <= $$maxval)>\
```

```

<$_if ($$_currbase is $$FM_File[$$val])>\
  <p class="SidebarTxt"><$$SideTitle[$$val]></p>\
<$_else>\
  <p class="SidebarTxt"><a class="SidebarLnk"\
    href="<$$FM_File[$$val]>.htm"><$$SideTitle[$$val]></a></p>\
  <$_endif>
<$$val++>
<$_endwhile>

```

28.4.5 Using pointers to process lists

The method described in §28.4.4 [Using macros to process lists](#) on page 807 is fine for a single pair of lists. But what if you have many such pairs of lists? Using the actual names of the lists in the macro means including as many copies of the [Sidebar] macro as there are pairs of lists, even though the functionality is identical for all pairs. Instead, you can construct a macro that works for every pair of lists, using *pointers* (indirect references) to the lists instead of the literal names of the lists.

To create a pointer to a list, assign the list name, in quotes, to a macro variable:

```
<$$ptr="$$list">
```

A set-up macro could initialize the pointers, starting index, and ending index for the lists, then invoke the [Sidebar] macro, which is now generalized for any pair of lists:

```

[SetupMySidebar]
<$$fileptr="$$FM_File">
<$$textptr="$$SideTitle">\
<$$val=2><$$maxval=6>
<$$Sidebar>

[Sidebar]
<$_while ($$val <= $$maxval)>\
  <$_if ($$_currbase is *$$fileptr[$$val])>\
    <p class="SidebarTxt"><*$$textptr[$$val]></p>\
  <$_else>\
    <p class="SidebarTxt"><a class="SidebarLnk"\
      href="<*$$fileptr[$$val]>.htm"><*$$textptr[$$val]></a></p>\
    <$_endif>
  <$$val++>
<$_endwhile>

```

The asterisks in front of `*fileptr[$$val]` and `*textptr[$$val]` indicate that these list variables are actually being used indirectly, as pointers to other lists. When you use the form `*$ptr[$$index]`, **Mif2Go** converts the reference internally to `$$list[$$index]` before retrieving the value.

*Process a list of
pointers*

If you need to access a list of pointers, do it in two steps. Suppose you have a list that contains pointers to the other two lists in the sidebar example:

```

[PtrList]
1=$$FM_File
2=$$SideTitles

```

To access an item in [FM_File], first assign to a macro variable the pointer-list item that points to [FM_File]:

```
<$$ptr=$$PtrList[1]>
```

Thereafter you can use the macro variable as a pointer to [FM_File]; and so, referring to the [FM_File] list in §28.4.4 [Using macros to process lists](#) on page 807, `<*$ptr[2]>` gets you the second item in the list, `descript`. You do not use quotes around the value in this case, because you want the actual list item in `$ptr`, not a reference to the list.

28.4.6 Using a list instead of a conditional expression

Suppose you want a different navigation bar for some of your HTML output files, depending on the name of the chapter from which the files are generated. One way would be to use a conditional expression (see §28.6.4.2 [Using conditional expressions](#) on page 815) to check the current chapter file name and choose the code for the navigation bar. For example:

```
[NavBar]
; Configure navigation bar for roadmap:
<$_if ($$_currbase is "user_roadmap")> <$rmap>
; Configure navigation bar for Programmer's Guide topics:
<$_elseif ($$_currbase is "bgp_user")> <$pgnav>
<$_elseif ($$_currbase is "mld_user")> <$pgnav>
... (long list of similar clauses)
<$_elseif ($$_currbase is "pga_user")> <$pgnav>
; Configure navigation bar for function topics, by default:
<$_else>
  <p>
    <a href="functions.htm">Function Index</a>
  </p>
<$_endif>
```

Instead, you could use a list indexed by the value of `$_currbase`, with each list value a macro call (or HTML code):

```
[navmap]
0 = <p><a href="functions.htm">Function Index</a></p>
user_roadmap = <$rmap>
bgp_user = <$pgnav>
mld_user = <$pgnav>
...
pga_user = <$pgnav>

[NavBar]
<$navmap[ $_currbase ]>
```

The 0 (zero) list item corresponds to the `<$_else>` clause in the original `[NavBar]` macro, and is used if the specified index (the value of `$_currbase`) is not found. Instead of a macro call the value of this list item is straight HTML code, which works as long as the code is all on one line. You could just as well use a macro call for the zero value, like the rest of the list items.

28.5 Accessing settings with configuration macros

You can access or change the current value of a configuration setting with a configuration macro that specifies a configuration variable.

§28.5.1 [Understanding configuration macros and variables](#) on page 809

§28.5.2 [Determining the value of a configuration variable](#) on page 810

§28.5.3 [Deploying configuration macros](#) on page 810

28.5.1 Understanding configuration macros and variables

A *configuration variable* is a macro variable that looks like this:

```
$$[Section]Key
```

where the components of the variable are as follows:

Section Name of a configuration-file section.

Key Keyword, format name, or other identifier that appears to the left of the equals sign in a configuration setting under [Section].

A *configuration macro* is a **Mif2Go** macro that employs a configuration variable, either to access or to change the value of a configuration setting.

This is all you need to access the current value of a configuration setting. To *change* the value of a setting, see §33.2.4 [Assigning values to configuration variables](#) on page 922.

Note: Specifying a configuration value with `$$_Section[Key]` as a predefined list variable is deprecated, though still supported for backward compatibility.

28.5.2 Determining the value of a configuration variable

The value of a configuration variable depends on whether the referenced setting is present and valid:

For present settings, value is the latest override (if any)

For missing settings, value is the default

For invalid settings, value is zero

For present settings, value is the latest override (if any)

The value of a configuration variable is the value of the setting in question at the time a macro is executed. If the original setting in your configuration file was overridden by a configuration-variable assignment in a marker or another macro, the override, not the original value, is the value returned for `<$$[section]key>`. See §33.2 [Overriding settings with markers or macros](#) on page 920.

For missing settings, value is the default

If you use a configuration variable to retrieve the value of a setting when the key is not present in your configuration file, or the section itself is missing from your configuration file, the value of `<$$[section]key>` is the default value specified for that key.

In some cases the default value is an empty string, as for a missing [Style*Prefix] or [Style*Suffix] setting.

For invalid settings, value is zero

If you use a configuration variable to retrieve a value when `<$$[section]key>` refers to an invalid configuration-file section, or to an invalid key, **Mif2Go** returns 0 (zero), which is interpreted as *false* in a conditional expression `<$_if($$[section]key ...)>`; see §28.6.4.2 [Using conditional expressions](#) on page 815.

28.5.3 Deploying configuration macros

To test the current value of a configuration setting (for example):

```
<$_if($$[HTMLOptions]ExtractEnable) ... >
```

To temporarily alter the value of a configuration setting (for example, to strip all table-specific HTML tags from format *Unruled* tables, but not from other tables):

```
[TableBeforeMacros]
Unruled = <$$[Tables]StripTable=1>
```

```
[TableAfterMacros]
Unruled = <$$[Tables]StripTable=0>
```

To convert only an individual graphic, leaving the rest unconverted:

```
[GraphStartMacros]
graphicID = <$$[Graphics]UseOriginalGraphicNames=0>
```

```
[GraphEndMacros]
graphicID = <$$[Graphics]UseOriginalGraphicNames=1>
```

To add a new setting with a configuration variable, see §33.2.5 [Adding a new configuration setting on the fly](#) on page 923.

28.6 Using expressions in macros

In **Mif2Go** macros, an *expression* usually consists of two operands separated by an operator:

```
<$(operand operator operand)>
```

As an exception, one type of conditional expression consists of three operands and two operators:

```
<$(operand ? operand : operand)>
```

The result of a **Mif2Go** macro expression is a strong value.

In this section:

- §28.6.1 [Understanding macro expressions](#) on page 811
- §28.6.2 [Understanding operands and operators](#) on page 811
- §28.6.3 [Displaying expression results in output](#) on page 813
- §28.6.4 [Using control structures in expressions](#) on page 815
- §28.6.5 [Specifying substrings in expressions](#) on page 817
- §28.6.6 [Using list variables in expressions](#) on page 818
- §28.6.7 [Using indirection in expressions](#) on page 819
- §28.6.8 [Removing spaces from strings: an example](#) on page 820

28.6.1 Understanding macro expressions

Result is a string value An expression always generates a string value, which for some purposes can be treated as a decimal integer number (or a hexadecimal number, depending on the operands); that is, you can do arithmetic on the result.

Decimal vs. hexadecimal The *numeric* result of an expression is decimal by default, unless the left operand is in hexadecimal format; then the result is in hexadecimal. You can coerce output to the other base by adding zero as the first term, expressed in the desired base, to the left operand. For example, you can coerce output to decimal with `(0 + 0x30)`, which yields 48; or to hexadecimal with `(0x0 + 31)`, to get `0x1F`.

Mif2Go does not support octal numbers or floating-point numbers.

Anonymous expressions Where you want to use the result of an expression, but you do not need to store the result for later use, you can use “anonymous” expressions; for example:

```
<$(($_count + 2)>
```

28.6.2 Understanding operands and operators

Operands for macro expressions An operand can be any of the following:

- a macro variable: `$$name` (see §28.3 [Using macro variables](#) on page 795)
- a number, including hexadecimal numbers starting with `0x` or `0X`
- a double-quoted string: `" . . . "`
- a single-quoted string: `' . . . '`
- a single-quoted character, including `'\r'` and `'\n'`
- an unquoted single word
- a parenthesized expression.

*Operators for
macro
expressions*

Operators include essentially the whole C-language numeric and logical sets, as well as some **Mif2Go** string operators; [Table 28-4](#) shows the operators you can use in macro expressions. Most operators participate in binary (two-operand) expressions. Exceptions are the operators used in the ternary conditional expression described in §28.6.4.2 [Using conditional expressions](#) on page 815, and the unary string operators described in §28.6.5 [Specifying substrings in expressions](#) on page 817.

Table 28-4 Operators for HTML macro expressions

Type	Operator	Meaning	Comments
Relational	<code>=, ==</code>	equal to	The result is 0 (zero) or 1 (one). Spaces are optional; you can use any number of spaces around symbol operators.
	<code>!=, <></code>	not equal to	
	<code><</code>	less than	
	<code><=</code>	less than or equal to	
	<code>></code>	greater than	
	<code>>=</code>	greater than or equal to	
Logical	<code>and, &&</code>	both operands are true	The result is 0 (zero) or 1 (one). Two-word operators must have exactly one space between the two words. You can use any number of spaces elsewhere. Unlike in C, both operands are always fully evaluated.
	<code>and not, &&!</code>	first is true, and second is false	
	<code>or, </code>	either is true, or both are true	
	<code>or not, !</code>	first is true, or second is false	
	<code>xor, ^</code>	one operand is true, the other is false	
	<code>xor not, ^!</code>	both are true, or both are false	
Bitwise	<code>&</code>	1 where both operands have 1 0 everywhere else	These are numeric string operators. The first six are like the logical operators. The last two are bitwise shifts with the second operand the count. For example: <code>((\$myvar >> 8) & 0xFF)</code> extracts the second-up byte from a number.
	<code>&~</code>	1 where first has 1 and second has 0 0 everywhere else	
	<code> </code>	1 where either has or both have 1 0 where both operands have 0	
	<code> ~</code>	1 where first has 1 or second has 0 0 where first has 0 and second has 1	
	<code>^</code>	1 where operand bits differ 0 where operand bits are the same	
	<code>^~</code>	1 where operand bits are the same 0 where operand bits differ	
	<code><< N</code>	shift first operand to the left <i>N</i> bits	
	<code>>> N</code>	shift first operand to the right <i>N</i> bits	
Arithmetic	<code>+</code>	plus	These are the usual suspects. The result of <code>(n / 0)</code> or <code>(n % 0)</code> is 0 (zero), because infinity is hard to represent.
	<code>-</code>	minus	
	<code>*</code>	times	
	<code>/</code>	divided by	
	<code>%</code>	modulo	

Table 28-4 Operators for HTML macro expressions (continued)

Type	Operator	Meaning	Comments
String	is	equal to	is and is not are caseless compares using <code>strcmp()</code>
	is not	not equal to	
	plus	concatenated with	plus is like <code>strcat()</code>
	before	substring before the 1st (leftmost) occurrence of 2nd string in 1st	before and after use <code>strstr()</code> to find the 2nd operand in the 1st: (doggie before gi) is dog
	after	substring after the first (leftmost) occurrence of 2nd string in 1st	You can get a <code>strnicmp()</code> effect using "first <i>N</i> " or "last <i>N</i> " with "is" or "is not": ((\$myvar first 3) is (\$yourvar last 3))
	first <i>N</i>	leftmost <i>N</i> characters (default = 1)	
	last <i>N</i>	rightmost <i>N</i> characters (default = 1)	
	length	length in characters	Integer result
	starts <i>\$\$str</i>	true if <i>\$\$str</i> is at the start	Boolean result
	ends <i>\$\$str</i>	true if <i>\$\$str</i> is at the end	Boolean result
	contains <i>\$\$str</i>	true if <i>\$\$str</i> occurs anywhere in the string	Boolean result
	char <i>N</i>	<i>N</i> th character, counting from left	First (leftmost) character is number 1 Default value of <i>N</i> is 1
	trim first <i>N</i>	all but first <i>N</i> characters	Default value of <i>N</i> is 1
	trim last <i>N</i>	all but last <i>N</i> characters	Default value of <i>N</i> is 1
	<i>\$\$str</i> lower	converts <i>\$\$str</i> to lowercase	
	<i>\$\$str</i> upper	converts <i>\$\$str</i> to uppercase	
	<i>\$\$str</i> replace <i>\$\$str1</i> with <i>\$\$str2</i>	converts each instance of <i>\$\$str1</i> in <i>\$\$str</i> to <i>\$\$str2</i>	
Conditional	?	"if" the 1st operand is true, "then" the 2nd operand is the value of the expression	<code><\$((\$myvar ? "yes" : "no")></code> is equivalent to:
	:	"else" the 3rd operand is the value of the expression	<code><\$_if (\$myvar)> yes <\$_else> no <\$_endif></code>

28.6.3 Displaying expression results in output

In general, **Mif2Go** macro expressions produce output. The exceptions are as follows:

- *assignments*, where much of the time you are going to use the assigned value later (see §28.3.2 [Assigning values to macro variables](#) on page 797)
- *control statements*, which have no obvious meaning of their own (see §28.6.4 [Using control structures in expressions](#) on page 815).

To display (that is, to include in HTML output) the result of evaluating an expression, enclose the expression in parentheses, as follows:

```
<$(... expr ...)>
```

You can also specify a display format to use, with **as** plus a C-language-style format string:

```
<$(... expr ...) as format-string>
```

A format string starts with “%” (percent sign) and is composed as follows, where any component enclosed in [] is optional:

```
%[flag(s)][width][.precision]format-code
```

The components of the format string can have any of the values listed in [Table 28-5](#).

Integer precision Suppose you wish to display the integer value of user variable `$$myint`, which you have set to internal value 5:

```
<$$myint = 5>
```

When you use a format string to display the value, the default integer precision is 1, as you can determine by comparing the results of the following expressions:

```
<$$myint as %0d>
<$$myint as %0.1d>
<$$myint as %0.3d>
```

The first two yield identical results, 5, while the third yields 005. However, when you do *not* use the “as %” construct, there is no precision; you get the internal string representation, which has three digits, unless you initialized it otherwise.

Table 28-5 Format components for displaying expression results

Component	Value	Effect on output
<i>flag</i>	-	The result is left justified in the display field (the default is right justified)
	+	The sign of the result is displayed (the default is to display the sign only for negative values)
	(blank)	A blank is displayed for positive values, a minus sign for negative values
	#	Hexadecimal result: displayed with the prefix 0X or 0x, depending on the format code Fractional decimal result: the decimal point is displayed Integer decimal result: no effect
<i>width</i>	(integer)	Minimum size of display field in characters
<i>precision</i>	(integer)	Integer result: minimum number of digits displayed (the default is 1) Fractional result: number of digits displayed after the decimal point String result: maximum number of characters displayed (the default is the entire string)
<i>format-code</i>	c	The result is displayed as a character
	d	The result is displayed as a decimal number
	s	The result is displayed as a string of characters
	x	The result is displayed as a hexadecimal number, with lowercase a through f
	X	The result is displayed as a hexadecimal number, with uppercase A through F

Additional format options For more information about C-language format strings and for additional components and format codes, see the following reference:

http://www.acm.uiuc.edu/webmonkeys/book/c_guide/2.12.html#printf

You can use any C-language format codes except those for floating-point values (e, f, g) or for pointers (p). It is best not to use the h or l (lowercase L) modifiers; however, if you ignore this advice, l is at least harmless.

As an example, this macro generates an ASCII table:

```
[Charset]
<$$cval = ' '>
<$_while ($$cval < '~')>\
<p><$$cval = ($$cval + 1) as %0.3d> \
0x<$$cval as %02X> \
<$$cval as %c></p>
```

Hexadecimal
output

In an expression, hexadecimal numbers beginning with 0x or 0X are understood as numeric values. But if you have a hexadecimal number stored in a variable, and try to display it like this:

```
<$$myvar as %d>      (or “as %c”, or even “as %x”)
```

you get 0 (zero) as output—for any hexadecimal number. Use the default (“as %s”), or just plain <\$\$myvar>.

28.6.4 Using control structures in expressions

Mif2Go provides predefined loop and conditional control structures for use in macro expressions. You cannot nest loop or conditional structures; instead, the outer macro must invoke another macro for the inner loop or conditional test.

In this section:

§28.6.4.1 [Understanding control-structure elements](#) on page 815

§28.6.4.2 [Using conditional expressions](#) on page 815

§28.6.4.3 [Using loop structures](#) on page 816

28.6.4.1 Understanding control-structure elements

The names of control-structure elements look almost identical to macro names. Avoid defining any macro of your own that has the same name as one of the control-structure elements listed in [Table 28-6](#); the **Mif2Go** control-structure definition takes precedence.

Table 28-6 Predefined control-structure elements

Control element	Where used	Purpose	Ref.
<\$_break>	Loop structure	Skip to the end of a loop	28.6.4.3
<\$_continue>	Loop structure	Jump back to the start of the next iteration	28.6.4.3
<\$_else>	Conditional expression	Introduce a final alternate condition	28.6.4.2
<\$_elseif>	Conditional expression	Introduce an intermediate alternate condition	28.6.4.2
<\$_endif>	Conditional expression	End a conditional expression	28.6.4.2
<\$_endrepeat>	Loop structure	End a count-down loop	28.6.4.3
<\$_endwhile>	Loop structure	End a logical loop	28.6.4.3
<\$_if>, <\$_if not>	Conditional expression	Begin a conditional expression	28.6.4.2
<\$_repeat>	Loop structure	Begin a count-down loop	28.6.4.3
<\$_until>	Loop structure	Begin a logical loop	28.6.4.3
<\$_while>	Loop structure	Begin a logical loop	28.6.4.3

28.6.4.2 Using conditional expressions

A conditional expression starts with:

```
<$_if (expr)>
```

or:

```
<$_if not (expr)>
```

and continues with:

```
<$_elseif (expr)>      (as many as you please)
```

```
<$_elseif not (expr)> (as many as you please)
```

```
<$_else>              (evaluated when no expr is true)
```

`<$_endif>` (optional if at the end of a macro)

(As an alternative to a long list of `<$_elseif>` clauses, you could use an indexed array for the *(expr)* values; see §28.4.6 [Using a list instead of a conditional expression](#) on page 809.)

Result of testing a string value

If *(expr)* has a string value, that value is seen as a non-number, and *(expr)* would evaluate to zero; that is, false. The relational operators always return “0” (false) or “1” (true), so if you had a variable `$$myword` with yes/no values, you would have to test the value like this:

```
<$_if ($$myword is yes)>
```

or like this

```
<$_if ($$myword is "yes")>
```

because, by itself:

```
<$_if ($$myword)>
```

would never be true.

How to nest conditionals

You cannot nest `<$_if>`s (and `<$_if not>`s) in the same macro; instead, call a second macro from within the first, and include the subordinate `<$_if>` (or `<$_if not>`) in the second macro. You specify a limit to such macro nesting with the following setting (see §28.1.3 [Nesting macros](#) on page 791):

```
[Macros]
MacroNestMax=128
```

Conditionals within expressions

You can also use C-style ternary operators “?” and “:”, for a shorthand version of a conditional expression. For example:

```
<$($$myvar ? "yes" : "no")>
```

instead of:

```
<$_if ($$myvar)>yes<$_else>no<$_endif>
```

The ternary operators give you a natural way to use conditionals *within* an expression, which is otherwise impossible.

28.6.4.3 Using loop structures

Mif2Go supports “while” loops, “repeat” loops, and “until” loops:

```
<$_while (expr)>...<$_endwhile>    loops while expr is not 0 (zero)
```

```
<$_repeat (expr)>...<$_endrepeat>  loops for the count of expr
```

```
<$_until (expr)>...<$_enduntil>    loops while expr is false
```

“While” loops

For `<$_while>`, a runaway-prevention feature ends the loop after the maximum count specified in the following setting:

```
[Macros]
; WhileMax = maximum count for <$_while>, to prevent runaways; the
; current count can be accessed using predefined macro variable
; <$$_wcount>
WhileMax=128
```

Predefined variable `<$$_wcount>` contains the loop count, starting with 1 (one).

“Until” loops

Because “until” is really the same as “while not”, the `<$_while>` runaway-prevention limit also applies to `<$_until>` loops:

```
[HtmlOptions]
WhileMax=128
```

For example, a loop controlled by `<$_until (0)>` goes to the `WhileMax` limit, unless you include an effective `<$_break>`. Predefined variable `<$$wcount>` contains the loop count, starting with 1 (one).

“Repeat” loops For `<$_repeat>`, the runaway-prevention limit comes into play only if the count is set to zero:

```
[Macros]
; RepeatMax = maximum count for <$_repeat> when value is not given, so
; that loop continues until a <$_break condition> is met
RepeatMax=128
```

The current loop count is held in predefined variable `<$$_count>`, and the down-count starting with `expr` is in predefined variable `<$$_dcount>`.

Nest loops You cannot nest a `<$_while>` in a `<$_while>`, or a `<$_repeat>` in a `<$_repeat>`, in the same macro. Nor can you nest a `<$_while>` in an `<$_until>`, or an `<$_until>` in a `<$_while>`. Instead you can call another macro to run a sub-loop. However, you can nest a `<$_while>` in a `<$_repeat>`, or a `<$_repeat>` in a `<$_while>`, and you can use one layer of `<$_if>` in the mix:

```
<$_while (expr)>
  <$_if (expr)>
    <$_repeat (expr)>...<$_endrepeat>
  <$_else>
    <$_repeat (expr)>...<$_endrepeat>
  <$_endif>
<$_endwhile>
```

Move around within loops You can use `<$_break>` to skip to the end of the loop, and `<$_continue>` to jump back to the start of the next iteration. Although you can invoke these control elements in `<$_if>`s, it is simpler to use the following constructs:

```
<$_break if (expr)>
<$_continue if (expr)>
```

Use only a single space before each `if`, and a minimum of one space after each `if`. These constructs work even in nested `<$_while>` or `<$_repeat>` loops, where they apply to the innermost of the loops where they occur.

28.6.5 Specifying substrings in expressions

You can determine the number of characters in a macro variable, and use string operators to extract substrings from the value of the variable. [Table 28-7](#) lists several of the string operators and shows how they are used in macro expressions.

See also:

§28.6.2 [Understanding operands and operators](#) on page 811

[Table 28-4 Operators for HTML macro expressions](#) on page 812

Table 28-7 String operators in macro expressions

Operator	Macro expression	Result of expression	
		Type	Value
length	<code>(\$\$string length)</code>	Integer	Number of characters in <code>\$\$string</code>
char	<code>(\$\$string char N)</code>	String	<i>N</i> th character in <code>\$\$string</code> , counting from the left; the leftmost character is number 1
first	<code>(\$\$string first N)</code>	String	First <i>N</i> characters of <code>\$\$string</code>
last	<code>(\$\$string last N)</code>	String	Last <i>N</i> characters of <code>\$\$string</code>

Table 28-7 String operators in macro expressions (continued)

Operator	Macro expression	Result of expression	
		Type	Value
before	(\$\$string before \$\$str)	String	Substring that precedes the first (leftmost) occurrence of \$\$str in \$\$string
after	(\$\$string after \$\$str)	String	Substring that follows the first (leftmost) occurrence of \$\$str in \$\$string
starts	(\$\$string starts \$\$str)	Boolean	True if \$\$str is at the start of \$\$string
ends	(\$\$string ends \$\$str)	Boolean	True if \$\$str is at the end of \$\$string
contains	(\$\$string contains \$\$str)	Boolean	True if \$\$str occurs anywhere in \$\$string
trim first	(\$\$string trim first <i>N</i>)	String	All but the first <i>N</i> characters of \$\$string
trim last	(\$\$string trim last <i>N</i>)	String	All but the last <i>N</i> characters of \$\$string
replace with	(\$\$string replace " " with "_")	String	Each instance of first operand is replaced with second operand
upper	(\$\$string upper)	String	\$\$string is all uppercase
lower	(\$\$string lower)	String	\$\$string is all lowercase

For example, to trim off the first four characters of \$\$mystring:

```
<$$mystring = ($$yourstring trim first 4)>
```

If the value of \$\$yourstring is “makework”, the value of \$\$mystring would be “work”.

Implied value of second operand

If the second operand *N* is missing from an expression that uses one of the following operators, a value of 1 (one) is assumed for *N*:

```
char
first
last
trim first
trim last
```

For example, to select only the last character, you can omit the second operand:

```
<$$yourstring = ($$mystring last)>
```

If the value of \$\$mystring is “groceries”, the value of \$\$yourstring would be “s”.

28.6.6 Using list variables in expressions

You might want to generate lists, such as lists by level of elements above the current element (its “ancestors”); see §28.4 [Using multiple-value list variables](#) on page 806.

In an expression, the following construct:

```
($$_paratag in $$mylist)
```

returns the value of the index for the current paragraph format in <\$\$mylist>, or 0 (zero) if missing; and:

```
($$mylist[$$level] is $_paratag)
```

You can set the list item with a normal assignment:

```
<$$mylist[$$level] = $_paratag>
```

28.6.7 Using indirection in expressions

Suppose you assign a variable to another variable, as follows:

```
<$$myvar = $$other>
```

Then, if you subsequently use:

```
<$$myvar>
```

you get whatever contents the variable named `$$other` had at the time you assigned it to the variable named `$$myvar`. Suppose you specified the original assignment like this:

```
<$$myvar = "$$other">
```

Then, if you subsequently use:

```
<$$myvar>
```

all you get is the literal string “`$$other`”. If instead you use:

```
<*$myvar>
```

you get the *current* contents of the variable `$$other` (but if there were no variable named `$$other`, you would get just the literal string “`$$other`”).

The same thing works through multiple layers. If you use this series of assignments:

```
<$$myvar = "$$other">
<$$other = "$$whatever">
<$$whatever = "here">
```

then, subsequently, the contents of `<*$myvar>` is “`here`”, which is the same as the contents of `<*$other>`, or of `<*$whatever>`, or even of `<*$whatever>`.

Now if you set:

```
<$$other = "something">
```

then:

```
<$$myvar>    gives:  $$other
<*$myvar>    gives:  something
```

If next you set:

```
<*$myvar = "something else">
```

then:

```
<$$other>    gives:  something else
<$$myvar>    gives:  $$other
<*$myvar>    gives:  something else
```

If finally you set:

```
<$$other = "$$myvar">
```

then (oops!):

```
<$$other>    gives:  nothing
```

*Runaway-
prevention limit*

However, a built-in circular-reference counter saves you from the natural consequences of this last foolish assignment. The counter prevents indirection through more than 128 levels.

The top-level variable is like an envelope that can contain more nested envelopes; you continue opening them until you get to the letter (the contents). You can use indirection to recurse, to process variables and expressions, and so forth, down to a simple value, through whatever layers that takes.

28.6.8 Removing spaces from strings: an example

Suppose you need to remove spaces and apostrophes from a string value (such as a topic title), and replace each space with an underscore, sending the result to output. The following macro uses several macro expression features:

```
[NewString]
<$_repeat ($$OldString length)>\
  <$$char = ($$OldString char $$_count)>\
  <$_if ($$char is " ")>\_
    <$_elseif ($$char is not "'")><$$char>\
    <$_endif>\
  <$_endrepeat>\
```

28.7 Passing a parameter to a macro

You can pass a single parameter to a macro by enclosing the value of the parameter in parentheses. **Mif2Go** evaluates the parameter as an expression, and the result of the expression is captured in predefined macro variable `$$_macroparam`. You can reference `$$_macroparam` in the same macro, and if you need to keep it around, assign its value to another macro variable.

For example, suppose you have a pair of before-and-after macros that surround the body of content intended to be rendered as a note:

```
[NoteBefore]
<p class="notehead"><$$_macroparam></p>
<p class="notebody">
```

You could change the heading to reflect the severity level of the note by invoking the macro like this:

```
<$NoteBefore("Warning")>
```

or like this:

```
<$NoteBefore("Note")>
```

28.8 Debugging macros

By default, **Mif2Go** ignores undefined or blank macros and macro variables; they do not appear in the output. However, if you are debugging a macro process, you might want the names of undefined (possibly misspelled) macros or macro variables to be flagged. To make the name of any blank (or undefined) macro or macro variable appear in the output where the value of the macro or variable would normally appear, specify one or both of the following options:

```
[Macros]
; NameUndefinedMacros = No (default)
; or Yes (insert $macro name in output)
NameUndefinedMacros=Yes
; NameUndefinedMacroVars = No (default)
; or Yes (insert $$macrovar name in output)
NameUndefinedMacroVars=Yes
```

28.9 Deploying macros and macro variables

In this section:

§28.9.1 [Understanding where to use macros and macro variables](#) on page 821

- §28.9.2 [Invoking macros at predetermined points in output](#) on page 821
- §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822
- §28.9.4 [Converting a dictionary-style list to an HTML table](#) on page 824
- §28.9.5 [Assigning macros to graphics or tables for HTML](#) on page 827
- §28.9.6 [Redefining navigation macros in HTML](#) on page 827
- §28.9.7 [Using HTML Macro markers to invoke macros](#) on page 828
- §28.9.8 [Implementing drop-down text with macros](#) on page 828

28.9.1 Understanding where to use macros and macro variables

You can use a macro to insert HTML or RTF code in any of the following places:

- before, after, or in place of:
 - a paragraph or character format
 - a graphic or a group of graphics
 - a table or a group of tables
- within table cells
- at any point in the text, using an **HTML Macro** or **Code** marker
- at fixed points in an HTML file, such as <head>, or at start and end of <body>
- at fixed points in an RTF file, such as in the header or footer, or at top or bottom.

You can give a macro variable an initial value in [MacroVariables] (see §28.3.2 [Assigning values to macro variables](#) on page 797); however, you can use macro variables only in the following two contexts:

- within **Mif2Go** macros
- within system commands (see §34.4 [Executing operating-system commands](#) on page 937).

Configuration settings whose values are not themselves macros cannot include macro variables.

28.9.2 Invoking macros at predetermined points in output

You can specify macros to be invoked at several predetermined points in HTML or RTF output, by assigning the macros to keywords. Locations for macro insertion depend on the type of output:

[HTML macro insertion points](#)

[RTF macro insertion points for Word](#)

[RTF macro insertion points for WinHelp](#)

*HTML macro
insertion points*

To insert macros in HTML output:

```
[Inserts]
; location = macro to insert, can call another macro
; TopicBreak is placed between topics when files are not split
; Entities is placed before the head element
; Head is placed after the title element within the head element
; Frames is placed between the head and body (for Framesets)
; Top is placed at the beginning of the body element
; Bottom is placed just before the ending of the body
; End is placed after the ending of the body (to close noframes)
```

For split and extract files, you can use variants of the [Inserts] keywords to restrict the types of files to which an inserted macro should apply; see §18.5.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 599.

For example:

```
[Inserts]
Top=<$EscapeFrameset>

[EscapeFrameset]
<SCRIPT LANGUAGE="JavaScript">
<!-- Begin
if (self.top.frames.length != 0)
    self.top.location=self.location;
// End -->
</SCRIPT>
<a target="_top"></a>
```

*RTF macro
insertion points
for Word*

To insert macros (or other content) in RTF output for Word:

```
[Inserts]
; location = content to insert, which may be a Mif2Go macro
; Top, Bottom
; Header, Footer
; FirstHeader, FirstFooter
; LeftHeader, LeftFooter
; RightHeader, RightFooter
```

*RTF macro
insertion points
for WinHelp*

To insert macros (or other content) in RTF output for WinHelp:

```
[Inserts]
; location = content to insert, which may be a macro
; TopicStart, TopicEnd
; SlideStart, SlideEnd
```

28.9.3 Surrounding or replacing text with code or macros

To specify code to be invoked before, after, or in place of a paragraph or character format:

1. List the format name in the configuration section appropriate for your output type:

```
[HTMLParaStyles] or
[HTMLCharStyles] for HTML, XML, or HTML-based Help
[WordStyles] for Word
[HelpStyles] for WinHelp.
```

2. Assign to the format one of the Code* properties listed in [Table 28-8](#). For example:

```
[HTMLParaStyles]
PopHead=CodeBefore
```

3. List the same format name in the corresponding [ParaStyleCode*] or [CharStyleCode*] or [AnumCode*] section, and assign to it whatever macros (or code, or both) you want inserted at that point in the resulting output. For example:

```
[ParaStyleCodeBefore]
PopHead=<$$isPopup=1>
```

This assignment can include any macros of the form <\$Macroname> or, for HTML output, <\$.\macrofile.htm>.

Note: If the code you assign requires more than one line, you *must* specify a macro for it, because a *key=value* entry cannot exceed one line; see §4.4 [Understanding the rules for configuration settings](#) on page 102.

You can also assign a macro to a format in section [StyleLinkSrc], to provide code for the href attribute of HTML links; see §19.2.4 [Specifying link properties with macros](#) on page 612.

Table 28-8 Macro code placement properties

Property	Configuration section*	HTML code placement	RTF code placement
CodeBefore	[ParaStyleCodeBefore], [CharStyleCodeBefore]	Before the starting element tag, such as <p>	Before the paragraph starting \pard, or before the opening brace for character formats
CodeAfter	[ParaStyleCodeAfter], [CharStyleCodeAfter]	Right after the closing element tag	Right after the closing \par, or after the closing brace for character formats
CodeBeforeAnum	[AnumCodeBefore]	Before the paragraph autonumber (does not apply to character formats)	Before the paragraph autonumber (does not apply to character formats)
CodeAfterAnum	[AnumCodeAfter]	After the paragraph autonumber (does not apply to character formats)	After the paragraph autonumber (does not apply to character formats)
CodeStart	[ParaStyleCodeStart], [CharStyleCodeStart]	Right after the starting element tag	At the start of the text, after properties; if a starting character format also has a CodeStart macro, both are used
CodeEnd	[ParaStyleCodeEnd], [CharStyleCodeEnd]	Before the closing element tag, such as </p>	At the end of the text just before \par, or before the closing brace for character formats
CodeReplace	[ParaStyleCodeReplace], [CharStyleCodeReplace]	Instead of paragraph or character content; any CodeBefore or CodeAfter is ignored	Instead of paragraph or character content; any CodeBefore or CodeAfter is ignored
LinkSrc	[ParaStyleLinkSrc], [CharStyleLinkSrc]	In the href attribute of an HTML link	Does not apply to RTF output

* For HTML conversions, **Mif2Go** recognizes section names prefixed with `Html` (as in `[HtmlStyleCodeAfter]`) for backward compatibility.

For example, to precede each major heading in HTML with an image:

```
[HTMLParaStyles]
Heading1=CodeBefore

[ParaStyleCodeBefore]
Heading1=<p class="MyImageTag"><$Mona></p>
```

Later in the configuration file, or in a macro library file:

```
[Mona]

```

The effect in the resulting HTML would be to display the image `smile.jpg` just before each element mapped from a *Heading1* paragraph in your FrameMaker document.

A macro does not have to be well formed by itself; only the end result must be well formed, after all macros are included. For example, suppose you use formats *A*, *B*, and *C*, one after the other, and you want all of them centered in HTML output. You could use these settings to achieve that effect:

```
[HTMLParaStyles]
A=CodeBefore
C=CodeAfter

[ParaStyleCodeBefore]
A=<div align="center">

[ParaStyleCodeAfter]
C=</div>
```

Text properties for RTF

You can use `[ParaStyleCodeBefore]` and `[ParaStyleCodeAfter]` to place ruled lines or images before and after a heading in RTF output. You can use `[ParaStyleCodeStart]` to add properties to text in RTF output, such as borders or background shading.

Tables for HTML You can use `[ParaStyleCodeBefore]` and `[ParaStyleCodeAfter]` to construct a table around a paragraph for HTML, possibly with an image in a cell; this works well for notes or tips. You can also construct a table around a series of paragraphs; see §28.9.4 [Converting a dictionary-style list to an HTML table](#) on page 824.

Entire document for HTML You could have a FrameMaker file that contains only a single paragraph, specify `CodeReplace` for that paragraph format, and assign to it a `[ParaStyleCodeReplace]` macro; then build the whole HTML output from macros, using macro variables (see §28.3 [Using macro variables](#) on page 795) to include specific content based on user entries.

See also:

§28.9.4 [Converting a dictionary-style list to an HTML table](#) on page 824

§28.3.7 [Creating macro variables from paragraph content](#) on page 802

28.9.4 Converting a dictionary-style list to an HTML table

Here is an example of combining configuration settings, macros, and macro variables to convert a dictionary-style list to a table in HTML.

In this section:

§28.9.4.1 [Stating the problem](#) on page 824

§28.9.4.2 [Assembling the pieces](#) on page 825

§28.9.4.3 [Putting it all together](#) on page 826

§28.9.4.4 [Making it work everywhere](#) on page 826

28.9.4.1 Stating the problem

Suppose your FrameMaker document contains dictionary-style lists in which the two columns of the list are implemented with a run-in paragraph format. For example:

AL[14:0]	= EEPROM word address
R0	= P/X buffer address
	<i>Note:</i> Required if X or P used
R6	= Y buffer address
SFVAR	= Memory bank selected:
	• EELIB_BANK_SELECT_X
	• EELIB_BANK_SELECT_Y
	• EELIB_BANK_SELECT_P
D_AUX_REG4	= Word count
CHECK_SUM	= Starting checksum

Suppose the formats used in such a list are as follows:

Type	Format name	Used for
Paragraph	<i>List_Term</i>	Dictionary terms (left column, run-in heading)
	<i>List_Defn</i>	Dictionary definitions (right column, run-in body)
	<i>Defn_note</i>	" <i>Note</i> " paragraph following a <i>List_Defn</i> * paragraph
	<i>Defn_Bullet</i>	Bullet items following a <i>List_Defn</i> * paragraph
	<i>Defn_Bullet_Last</i>	Last bullet item following a <i>List_Defn</i> * paragraph
Character	<i>Italics</i>	Autonumber for <i>Defn_note</i> paragraphs
	<i>CodeStyle</i>	Any code content in a <i>List_Defn</i> * or <i>Defn</i> * paragraph

Next, see §28.9.4.2 [Assembling the pieces](#) on page 825.

28.9.4.2 Assembling the pieces

To have **Mif2Go** convert a list that uses a run-in format in FrameMaker (as described in §28.9.4.1 [Stating the problem](#) on page 824) to an HTML table, you must provide configuration settings and HTML code to meet the following challenges:

[Coping with a run-in format](#)
[Starting and ending the table](#)
[Starting and ending each row](#)
[Starting and ending each cell.](#)

Coping with a run-in format

By default, **Mif2Go** treats run-in paragraph formats more like character formats. This means that macros you want inserted before the *List_Term* items would be embedded in the start of the *List_Defn* paragraphs instead (which is what you would want if you were trying to preserve the run-in effect in HTML instead of converting the list to a table). To get macros in the right place around *List_Term* paragraphs, you would need to set the following option:

```
[HTMLOptions]
RunInHeads=Normal
```

See §21.3.2 [Converting sidehead and run-in paragraph formats](#) on page 648.

A more involved alternative would be to use a conversion template to replace the run-in formats with regular paragraph formats; see §3.4.1 [Importing formats from a FrameMaker template](#) on page 79.

Starting and ending the table

If your document has only one such dictionary-style list, you could insert **Code** markers containing macros to provide the `<table>` and `</table>` tags; see §29.7 [Inserting code or text with markers](#) on page 842. However, if your document contains many such lists, instead you would want to use macros that employ the following:

- macro variables (see §28.3.1 [Creating and invoking macro variables](#) on page 796)
- `<$_if>` conditions (see §28.6.4.2 [Using conditional expressions](#) on page 815).

For example, you could define and initialize a macro variable to keep track of where to start and end the table:

```
[MacroVariable]
InTable=0
```

You could start a *CodeBefore* macro for the run-in *List_Term* format with:

```
<$_if ($$InTable==0)><table attr=val ... >$$InTable=1><$_endif>
```

This code guarantees that the first *List_Term* paragraph in the list is preceded by an opening `<table>` tag, yet subsequent *List_Term* paragraphs are not, as long as the table is still being generated.

To identify the end of the list, you could assign a *CodeBefore* macro to any paragraph format that can *follow* the list (but never appear *within* the list):

```
<$_if ($$InTable==1)></table><$$InTable=0><$_endif>
```

Starting and ending each row

Each row of the table starts with a *List_Term* paragraph, and ends just before the next *List_Term* paragraph (or the end of the list); and the beginning and end of a row also mark the beginning of the first cell and end of the second cell, respectively. This means the *CodeBefore* macro for the *List_Term* format can provide row and cell tags to start and end each row, as well as start the first cell in the row and end the second cell:

```
<$_if ($$InTable==0)>
  <table attr=val ... >$$InTable=1>
<$_else>
  </td></tr>
```

```
<$_endif>
<tr><td>
```

*Starting and
ending each cell*

Each *List_Defn* paragraph marks the end of a *List_Term* cell and the beginning of a *List_Defn* cell (which might also include other paragraph formats following the *List_Defn* paragraph; see §28.9.4.1 [Stating the problem](#) on page 824). Therefore, a *CodeBefore* macro assigned to the *List_Defn* format can provide tags for the end of the first cell and beginning of the second cell:

```
</td><td>
```

Next, see §28.9.4.3 [Putting it all together](#) on page 826.

28.9.4.3 Putting it all together

To construct a two-column HTML table around the dictionary list (after §28.9.4.2 [Assembling the pieces](#) on page 825), you can use the following settings:

```
[HTMLParaStyles]
*=CodeBefore

[ParaStyleCodeBefore]
List_Term=<$TableStart>
List_Defn=<$BetweenCells>
Defn*=
Italics=
CodeStyle=
*=<$TableEnd>
```

Only the two formats that form the dictionary list need the `<$TableStart>` and `<$BetweenCells>` macros; other formats either do not need to be involved in constructing the table, or are not part of the list. Empty settings for the other paragraph and character formats used in the list prevent the table from ending prematurely. All other formats are assigned `<$TableEnd>`, which acts only when a table is being generated from a list.

The macros are defined as follows:

```
[TableStart]
<$_if ($$InTable==0)><table attr=val ... ><$$InTable=1>\
<$_else></td></tr><$_endif>\
<tr><td>

[BetweenCells]
</td><td>

[TableEnd]
<$_if ($$InTable==1)></td></tr></table><$$InTable=0><$_endif>
```

Next, see §28.9.4.4 [Making it work everywhere](#) on page 826.

28.9.4.4 Making it work everywhere

At this point (after §28.9.4.3 [Putting it all together](#) on page 826) you have most of the configuration settings, HTML code, and macro logic you need to create an HTML table from a dictionary-style list. Now you must make sure the table ends gracefully in all contexts.

The `<$TableEnd>` macro is defined with a condition, so that it does nothing unless a table is being generated from a dictionary list. This makes it safe to assign to any text format that is not part of a dictionary list. But what if the list is inside a table in FrameMaker, or is immediately followed by a table, or is at the very end of the file? You must make sure the generated table is “tied off” before tags are required for an enclosing

or following table, because those tags must *precede* any non-list paragraphs that would otherwise end the generated table.

You must meet the following challenges:

- End the table before starting another table
- End the table before ending an enclosing cell
- End the table if the file ends.

*End the table
before starting
another table*

To make sure the dictionary-list table ends before another table begins:

```
[TableBeforeMacros]
; TableID = macro to put before table start, top title or indent
a_table_group=<$TableEnd><DIV class="twide">
*=<$TableEnd>

[TableAfterMacros]
; TableID = macro to put after table end or bottom title
a_table_group=</DIV>
```

The call to `<$TableEnd>` comes before any other code you assign to a particular table (or table group); and before any other table. This ensures that the generated table ends before another table starts. You can insert the `<$TableEnd>` macro in several other places, to position it with respect to other tables; see §24.6.1 [Invoking macros around tables](#) on page 748.

*End the table
before ending an
enclosing cell*

To handle the case where the generated table is inside a cell of another table (in FrameMaker), you must invoke `<$TableEnd>` at the end of every FrameMaker table cell:

```
[TableCellEndMacros]
*=<$TableEnd>
```

This setting applies to the cells of an enclosing table in FrameMaker, not to HTML table cells that are being generated.

*End the table if
the file ends*

What if a list is at the end of a file, so there is no text, table, or other construct immediately following the generated table? You can invoke the same table-ending macro here:

```
[Inserts]
End=<$TableEnd>
```

See §18.5.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 599.

28.9.5 Assigning macros to graphics or tables for HTML

You can specify macros to be invoked before, after, or in place of a graphic, or a group of graphics, by assigning macros to a `GraphicID` in one of the `[Graph*Macros]` sections; see §23.5.2 [Replacing or surrounding a graphic with macro code](#) on page 710.

You can specify macros to be invoked before, after, or in place of a table, or a group of tables, by assigning macros to a `TableID` in one of the `[Table*Macros]` sections; see §24.6 [Using macros to control table properties](#) on page 748 and §24.6.1 [Invoking macros around tables](#) on page 748.

You can also specify macros to be invoked inside tables; see §24.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 750.

28.9.6 Redefining navigation macros in HTML

You can assign macros to keywords in the `[NavigationMacros]` section to redefine the behavior of Mif2Go-supplied browse macros `<$_prev>` and `<$_next>`. For example:

```
[NavigationMacros]
; PrevMacro = content to put out for <$_prev>
```



```

; NextMacro = content to put out for <$_next>
; PrevFSMacro = macro to use for <$_prev> at start of file
; NextFSMacro = macro to use for <$_next> at end of file
; StartingPrevFSMacro = <$_prev> to use at start of first file
; EndingNextFSMacro = <$_next> to use at end of last file

```

See §20.4 [Creating a browse sequence](#) on page 635 for information about using these and related settings.

28.9.7 Using HTML Macro markers to invoke macros

You can specify a macro to be invoked via the FrameMaker **HTML Macro** marker: the same marker type used by FrameMaker HTML export. Insert a marker of type **HTML Macro** where you want a macro to be invoked, and supply `<$Macroname>` or `<$.\macrofile.htm>` as the marker text.

For compatibility with existing FrameMaker HTML macro usage, you can also specify just the macro name in a FrameMaker **HTML Macro** marker, without the angle brackets and dollar sign; **Mif2Go** processes the marker text as though it were a **Mif2Go** macro.

28.9.8 Implementing drop-down text with macros

The following sections of the **Mif2Go User's Guide** present examples of macros that incorporate JavaScript to dynamically expand and collapse areas of text:

§7.9.7 [Deploying JavaScript code for drop-down sections](#) on page 234

§7.9.8 [Emulating Web Works Publisher drop-down hotspots](#) on page 237

You can modify these macros for your own purposes.

28.10 Using macros to fine-tune HTML or XML output

You can use macros and macro variables to solve special HTML or XML problems that are not addressed by the usual **Mif2Go** configuration settings. This section describes the best approach.

Start from the HTML end. Take one of the output .htm files **Mif2Go** generates from your FrameMaker document and use a plain-text editor to modify the HTML code:

1. Look at the code **Mif2Go** produces, and decide what additional bits of code are needed to achieve the effect you want; for example, `<table>`, `<tr>`, and `<td>` tags to create a two-cell table around an in-line image and its adjacent text.
2. Add the bits of code to the HTML, on lines of their own where possible, and view the result in a browser. You might have to experiment with variations until you get the effect you want.
3. Include the successful HTML code in **Mif2Go** macro definitions. Make each separate chunk of added code into one macro. For example (assuming the anchor for each in-line image is at the very start of the adjacent text):

```

[GrInfoBefore]
; Start a table, row, and cell just before an in-line image:
<table class="GrInfo"><tr><td>\

[GrImgEnd]
; After an in-line image, start a new cell for the adjacent text:
</p></td><td><p class="Body">

```

```
[GrInfoAfter]
; After the adjacent text, end the cell, row, and table:
</p></td></tr></table>
```

See §28.1.1 [Defining macros](#) on page 787.

4. Consider where your new HTML-code macros should go in the document flow. Do they precede the opening of some type of paragraph? Follow the closing? Go at the top or bottom of the page? Or just get plunked in at arbitrary points? You might have to define some new paragraph formats in FrameMaker to identify places to invoke the macros. For example, if sometimes you have multiple paragraphs of text adjacent to an in-line image, you might need three different format names for those paragraphs:

GrInfoStart — First paragraph

GrInfoEnd — Last paragraph

GrInfo — Sole paragraph

If you do not want to change format names, you could put **HTML Macro** markers before and after each instance of adjacent text. The starting marker would contain:

```
<$GrInfoBefore>
```

and the ending marker would contain:

```
<$GrInfoAfter>
```

You would keep the same macros defined in [Step 3](#), and get the same result.

5. Tell **Mif2Go** where to invoke the macros in the output, so the code gets inserted in the right places automatically, by adding settings to the configuration file to invoke your new macros. For example, to invoke the macros defined in [Step 3](#) whenever **Mif2Go** encounters paragraphs in the formats defined in [Step 4](#):

```
[HTMLParaStyles]
; Assign code placement to each GrInfo* paragraph format:
GrInfoStart=CodeBefore
GrInfoEnd=CodeAfter
GrInfo=CodeBefore CodeAfter

[ParaStyleCodeBefore]
; Starting and sole paragraphs need code just before them:
GrInfo*=<$GrInfoBefore>

[ParaStyleCodeAfter]
; Ending and sole paragraphs need code to follow them:
GrInfo*=<$GrInfoAfter>

[GraphEndMacros]
; The image itself needs code to close its cell:
*=<$GrImgEnd>
```

See §28.1.2 [Invoking a macro](#) on page 791 and §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

6. Convert the file again, and see if the new code shows up where it is needed. Does the code also pop up where it is not wanted? If so, you can include a test to prevent the code from appearing in other places. For example, to avoid creating a table around a graphic that does not have adjacent text, you could modify the macros in [Step 3](#) to use a macro variable and a conditional expression (both shown in **boldface**):

```
[GrInfoBefore]
<$$GrInf = 1>
<table class="GrInfo"><tr><td>\

[GrImgEnd]
<$_if ($$GrInf)></p></td><td><p class="Body"><$_endif>
```

```
[GrInfoAfter]
</p></td></tr></table>
<$$GrInf = 0>

[MacroVariables]
; Put any macro definition sections before this section.
GrInf=0
```

See §28.3 [Using macro variables](#) on page 795 and §28.6.4.2 [Using conditional expressions](#) on page 815.

29 Working with FrameMaker markers

FrameMaker markers provide a way to introduce HTML or RTF code and **Mif2Go** configuration overrides into your document, without affecting the original content or creating format dependencies. Topics include:

- §29.1 [Using custom FrameMaker markers](#) on page 831
- §29.2 [Adding custom marker types](#) on page 832
- §29.3 [Remapping marker types and hypertext commands](#) on page 836
- §29.4 [Defining and redefining marker behavior](#) on page 838
- §29.5 [Suppressing markers](#) on page 841
- §29.6 [Using marker property names for marker types](#) on page 842
- §29.7 [Inserting code or text with markers](#) on page 842
- §29.8 [Identifying markers with variable <\\$\\$_objectid>](#) on page 847

29.1 Using custom FrameMaker markers

You can add custom marker types in FrameMaker, and use configuration settings to define the behavior of markers of those custom types in **Mif2Go** conversions. You can also redefine the behavior of existing marker types and hypertext commands.

You can do the following with FrameMaker markers:

- **Add** custom marker types. **Mif2Go** uses custom markers to produce predefined effects; see §29.2 [Adding custom marker types](#) on page 832.
- **Invent** new marker types, and assign properties to them; see §29.3 [Remapping marker types and hypertext commands](#) on page 836.
- **Remap** most marker types, and optionally give them new properties; see §29.3 [Remapping marker types and hypertext commands](#) on page 836 and §29.4 [Defining and redefining marker behavior](#) on page 838.
- **Redefine** the behavior of most marker types; see §29.4 [Defining and redefining marker behavior](#) on page 838.
- **Map** custom markers directly to **Hypertext** marker subtypes (hypertext commands); see §29.3 [Remapping marker types and hypertext commands](#) on page 836.

For example, you can use custom markers to do any of the following:

- Designate split points and extract extents for HTML (see §18.2.1 [Designating split points](#) on page 586).
- Insert WAI markup (see §25 [Generating WAI markup for HTML](#) on page 755).
- Provide ALink entries for Help systems (see §7.6 [Providing related-topic links for Help systems](#) on page 219).
- Identify context-sensitive help targets (see §7.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 239).
- Assign alternate configuration values to individual tables, graphics, paragraphs, or character spans (see §33.2.2 [Overriding settings with configuration markers](#) on page 921).
- Add links and instructions (§34.1.2 [Using markers to add links and instructions](#) on page 935).

See also:

- §5.11 [Repurposing FrameMaker markers](#) on page 139

29.2 Adding custom marker types

To add a new marker type to your document, use the FrameMaker *Edit Custom Marker Type* dialog, reached via **Special > Marker > Marker Type: Edit...**

Note: Some custom marker types have dedicated purposes in Mif2Go conversions.

In this section:

§29.2.1 [Identifying dedicated custom marker types](#) on page 832

§29.2.2 [Naming new custom marker types](#) on page 834

§29.2.3 [Understanding attribute markers](#) on page 834

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

29.2.1 Identifying dedicated custom marker types

Each of the custom marker types listed in [Table 29-1](#) produces a predefined effect when Mif2Go encounters a marker of that type in your document.

Table 29-1 Custom marker types with predefined effects

Marker type	Purpose	Effect
ALink	Help-system associative link	Content is the text of an ALink identifier
ANSI	Character -set mapping for HTML	Select an alternate character-set table for mapping; see §13.4.3.2 Selecting a Windows code page for single-byte character sets on page 431
CellAttr	Table mark-up for HTML and XML	Content is the value of the HTML <td> tag or <th> tag attribute named by Attr , for the enclosing cell
CellClass	Table mark-up for HTML	Content names the CSS <code>class</code> for a table cell
CellGroup	Table mark-up for HTML	Content specifies the group of a header cell
CellID	Table mark-up for HTML	Overrides generated WAI ID attributes for a cell
CellScope	Table mark-up for HTML	Content specifies the scope of a header cell
CellSpan	Table mark-up for HTML	Assigns the <code>span</code> property to a header cell
Code	Insert HTML or RTF code	Content is used as code; macros are expanded
Config	Change configuration setting for HTML or RTF	Content is a <code>[Section]Key=Value</code> or <code>[Section]=Value</code> setting for HTML or RTF output
Delete	No standalone purpose	Deletes itself
DITA*	Provide DITA mark-up	See Table 15-3 on page 536
DocBook*	Provide DocBook mark-up	See Table 17-2 on page 583
EclipseAnchor	Merge Eclipse Help projects	Marks where a secondary TOC should be inserted
EclipseContext	Context-sensitive help	Content is the context ID for an infopop
EclipseLink	Merge Eclipse Help projects	Content includes path to secondary TOC file
ExtCodeEndChar	Include text from external files	Last character of external file to include
ExtCodeEndLine	Include text from external files	Last line of external file to include
ExtCodeFileEnc	Include text from external files	Encoding of external file
ExtCodeFileLen	Include text from external files	Length of external file in characters
ExtCodeStartChar	Include text from external files	First character of external file to include
ExtCodeStartLine	Include text from external files	First line of external file to include

Table 29-1 Custom marker types with predefined effects (continued)

Marker type	Purpose	Effect
ExtrBottom	Extract files for HTML	Content is the last item in the <body> of an extract
ExtrDisable	Extract files for HTML	Turns off extract processing
ExtrEnable	Extract files for HTML	Turns on extract processing
ExtrEnd	Extract files for HTML	Ends an extract
ExtrFinish	Extract files for HTML	Marks the last paragraph of an extract
ExtrHead	Extract files for HTML	Content is placed in the <head> of an extract
ExtrReplace	Extract files for HTML	Content replaces an extract in the original file
ExtrStart	Extract files for HTML	Marks the first paragraph of an extract
ExtrTop	Extract files for HTML	Content is the first item in the <body> of an extract
FileName	Split or extract HTML files	Content is the name of a split or extract file
GraphAttr	Image mark-up for HTML and XML	Content is the value of the HTML tag attribute named by Attr for the next image
GraphDpi	Image resolution	Overrides any other DPI setting for the graphic
HelpMerge	Merge Help files	Marks where the named Help file should be inserted
HTMLComment	Add comments to HTML	Content is the text of an HTML comment
HTMConfig	Change configuration setting for HTML	Content is a [Section]Key=Value or [Section]=Value setting for HTML
HVIndex	Index entries	Special index marker for Microsoft Help Viewer 1.x
JH2PopProp	JavaHelp window property	Content is a JavaHelp 2 pop-up window parameter
JH2SecProp	JavaHelp window property	Content is a JavaHelp 2 secondary window parameter
KeyrefBranch	Map branch processing	Names the map branch to use to resolve the next keyref
LinkAttr	Link mark-up for HTML and XML	Content is the value of the HTML tag attribute named by Attr , for the next link
LinkClass	Link mark-up for HTML	Content names the CSS class for the next link
LocalTOCTitle	Split files for HTML	Content is the text of a local-TOC link
MetaType	<meta> tag for HTML	Content is the content value for a new <meta name=Type content=...> tag
RowAttr	Table mark-up for HTML and XML	Content is the value of the <tr> or <row> attribute named by Attr , for the row of the enclosing cell
RTFConfig	Change configuration setting for RTF	Content is a [Section]Key=Value or [Section]=Value setting for RTF
Search	Conditional output	Content determines whether content is included in FTS
Split	Split files for HTML	Marks a split point in a FrameMaker file
TableAttr	Table mark-up for HTML and XML	Content is the value of the HTML <table> tag attribute named by Attr , for the enclosing table
Title	Split or extract files for HTML	Content is the page title of a split or extract file
TopicAlias	Context-sensitive help	Inserts a named CSH target in output
TopicStartCode	<head> code for HTML	Code is executed before topic content is processed
Window	HTML Help secondary window	Content names a window for jumps from contents or index

29.2.2 Naming new custom marker types

When you invent a new custom marker type in FrameMaker, the name must not conflict with your use of any of the predefined marker types listed in [Table 29-1](#).

Avoid adding a custom marker type whose name has any of the following characteristics:

- Duplicates the name of a marker type listed in [Table 29-1](#), if you intend to use markers of that type for the purpose shown; see §29.2.1 [Identifying dedicated custom marker types](#) on page 832.
- Duplicates the name of a property listed in [Table 29-3](#), unless you intend to use such markers for the stated purpose; the marker type takes on the named property. See §29.6 [Using marker property names for marker types](#) on page 842.
- Begins with **Cell**, **Char**, **Graph**, **Link**, **Meta**, **Para**, **Row**, or **Table** if you are generating HTML or XML, unless you end the name with a valid attribute name. All such markers are assumed by Mif2Go to be attribute markers; see §29.2.4 [Using attribute markers for HTML or XML](#) on page 835.
- Begins with **JH2Pop** or **JH2Sec** if you are generating JavaHelp 2, unless you end the name with a valid JavaHelp 2 window-access object property; see §11.8.1.5 [Overriding window-access properties with markers](#) on page 397.

29.2.3 Understanding attribute markers

An *attribute marker* includes the name of the attribute as a suffix to the predefined custom marker type name. The content of the marker becomes the value of the attribute for the applicable element tag:

```
<elementname attributetype="content">
```

For example, for HTML output, a **Rowbgcolor** marker with content yellow, placed in a FrameMaker table cell, would add the attribute bgcolor with value yellow to the HTML <tr> tag for the current table row:

```
<tr bgcolor="yellow">
```

Nonconforming attribute markers

A few attribute markers do not conform exactly to this naming and usage convention; for example, WAI support markers **CellGroup** and **CellSpan**. See §26.2.4 [Assigning table-cell attribute values with custom markers](#) on page 772. Another nonconforming attribute marker is **MetaType**. For HTML output, this marker causes a <meta> tag to be added to the <head> element; **Type** becomes the value of the name attribute, and the content of the marker becomes the value of the content attribute.

Concatenated attribute markers

Although the text of a FrameMaker marker is limited to 256 characters, Mif2Go gets around that restriction for attribute markers by concatenating all markers for the same attribute that are inserted before the next item to which they apply. You can just add more markers of the same type, and continue the content.

Also:

- Inserting another marker of a different type between two markers for the same attribute does not prevent concatenation, even if the middle marker is a different attribute marker for the same element.
- If you want the content of two concatenated attribute markers to be separated by a space in the attribute value, you must provide the space, either at the end of content in the first marker or at the beginning of content in the second marker.

Extra attributes

Using markers to add attributes can result in extra attributes for a given tag. Browsers ignore extra attributes, but validators would not be pleased; see §13.16 [Passing W3C validation tests](#) on page 453. (Of course validators would not be pleased with most of what is on the Web, so that might be of little consequence.)

Duplicate markers If multiple attribute markers with identical names but different content apply to the same element, **Mif2Go** uses the content of the last marker encountered as the value of the attribute.

See also:

§25.1.3 [Creating custom markers for WAI attributes](#) on page 756

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

§29.7.2 [Surrounding marker content with code](#) on page 843

29.2.4 Using attribute markers for HTML or XML

For HTML or XML output, **Mif2Go** treats any FrameMaker marker that has a name that begins with **Cell**, **Char**, **Graph**, **Link**, **Meta**, **Para**, **Row**, or **Table** as an attribute marker. For HTML (for example), **Mif2Go** inserts the `attribute="value"` pair specified by each of the attribute marker types as follows:

CellAttr In the `<td>` or `<th>` tag for the enclosing table cell.

CharAttr In the tag for the current or next inline element.

GraphAttr In the next `` tag.

LinkAttr In the next link (``) tag.

MetaType In a `<meta>` tag; produces a new element, `<meta name="Type" content="content">`, in the `<head>` element.

ParaAttr In the tag for the current block element.

RowAttr In the `<tr>` tag for the current table row; best practice is to place the marker in the first cell in the row.

TableAttr In the `<table>` tag, in the enclosing table; if not positioned in a table, applies to the next table in the same flow.

[Table 29-2](#) lists the elements to which each attribute marker can apply for each output type.

Table 29-2 Elements to which attribute markers apply, by output type

Marker	Output type			
	HTML/XHTML	Generic XML	DITA XML	DocBook XML
CellAttr	<code><td></code> , <code><th></code>	<code><td></code> , <code><th></code>	<code><entry></code> , <code><stentry></code> , <code><choption></code> , <code><chdesc></code> , <code><proptype></code> , <code><propvalue></code> , <code><propdesc></code>	<code><td></code> , <code><th></code>
CharAttr	<i>inline elements</i>	<i>inline elements</i>	<i>inline elements</i>	<i>inline elements</i>
GraphAttr	<code></code>	<code></code>	<code><image></code>	<code><imagedata></code>
LinkAttr	<code><a></code> ((does not apply to Help pop-ups, secondary window jumps, or footnote cross references)	<code><a></code> (applies to the <code>AtagElement</code> setting; do not use for name; overridden by <code>XMLLinkAttrs</code>)	<code><xref></code> (can add to or replace standard <code>href</code> , <code>type</code> , <code>format</code> , and <code>scope</code> attributes)	<code><xref></code> , <code><ulink></code>
MetaType	<code><meta></code>	<code><meta></code>		

Table 29-2 Elements to which attribute markers apply, by output type

Marker	Output type			
	HTML/XHTML	Generic XML	DITA XML	DocBook XML
ParaAttr	<i>block elements</i>	<i>block elements</i>	<i>block elements</i>	<i>block elements</i>
RowAttr	<tr>	<tr>	<row>, <strow>, <chrow>, <property>	<tr>, <row>
TableAttr	<table>	<table>	<table>, <simplatable>, <choicetable>, <properties>	<table>

29.3 Remapping marker types and hypertext commands

You can reuse the content of most FrameMaker markers, and also create new marker types, by remapping an existing marker type to one or more other marker types. You can also remap certain FrameMaker hypertext commands.

In this section:

§29.3.1 [Remapping and cloning marker types](#) on page 836

§29.3.2 [Understanding when to remap marker types](#) on page 837

§29.3.3 [Remapping FrameMaker hypertext commands](#) on page 837

29.3.1 Remapping and cloning marker types

To remap a marker type, and optionally clone the remapped type:

```
[Markers]
; marker type name = one or more marker type names
FM_Marker = FM_Marker ClonedMarker AnotherClonedMarker ...
```

Once you remap a marker type, the original marker type (to the left of the equals sign) is no longer in effect, unless you also specify its name to the right of the equals sign. For example:

```
[Markers]
Index = Index MySpecialIndex
```

This assignment retains use of the original FrameMaker **Index** markers, while also cloning them as new **MySpecialIndex** markers, to which you can assign other properties; see §29.4.1 [Assigning properties to marker types](#) on page 838.

You can remap the following FrameMaker marker types:

- Author**
- Comment**
- Cross-Ref**
- Equation**
- Glossary**
- Header/Footer \$1**
- Header/Footer \$2**
- HTML Macro**
- Hypertext**
- Index**
- Subject**

You must observe the following restrictions:

- The **Conditional Text** marker type cannot be remapped.
- Names of marker types you are remapping *to* (names to the right of the equals sign) may not contain spaces or commas (those to the left of the equals sign may contain spaces and commas).

You can remap any marker type (except **Conditional Text**) to:

- one or more existing or predefined (see [Table 29-1](#)) marker types or hypertext commands
- any new marker type(s) you name to the right of the equals sign
- itself, for the purpose of redefining the behavior of that marker type; see §29.4.1 [Assigning properties to marker types](#) on page 838.

29.3.2 Understanding when to remap marker types

Use `[Markers]` primarily to make new marker types from existing markers, and to clone the new markers. You might need to do this if you are making extracurricular use of FrameMaker **Index** markers, because you cannot redefine **Index** marker properties. See §14.8 [Converting index entries to generic XML](#) on page 468.

For example, to add all FrameMaker **Subject** markers to the index, and also make clones of the remapped markers as new marker type **ALinkRef**:

```
[Markers]
Subject = Index ALinkRef
```

For another example, to remap context-sensitive help targets identified with **TopicAlias** markers to hypertext **newlink** markers:

```
[Markers]
TopicAlias = newlink
```

Note: This mapping is no longer required for context-sensitive help. Do not use it in conversions to DITA XML, or the CSH targets will be omitted from output; see §15.14 [Including CSH targets in DITA XML](#) on page 535.

Because many marker types have special purposes or require specific content, be careful about remapping to custom marker types. For example, do not try to remap to attribute markers used for WAI support; see §25.1.3 [Creating custom markers for WAI attributes](#) on page 756 and §A [WAI marker library for HTML](#) on page 1013.

29.3.3 Remapping FrameMaker hypertext commands

To remap a FrameMaker hypertext command to a marker type, and optionally clone the marker type:

```
[Markers]
; hypertext command name = one or more marker type names
command = FM_Marker ClonedMarker AnotherClonedMarker ...
```

You can remap the following FrameMaker **Hypertext** marker commands:

<u>Hypertext command</u>	<u>FrameMaker Hypertext dialog command name</u>
alert	Alert
gotolink	Jump to Named Destination
gotolinkfitwin	Jump to Named Destination & Fit to Page
gotoObjectID	(None)
gotoObjectIDfitwin	(None)
gotopage	Jump to Page Number

<u>Hypertext command</u>	<u>FrameMaker Hypertext dialog command name</u>
message (for message URL)	Go to URL
newlink	Specify Named Destination
openlink	Open Document
openlinkfitwin	Open Document & Fit to Page
openObjectID	(None)
openObjectIDfitwin	(None)

Hypertext marker command **message** specifically means **message URL**. A marker of this subtype should have a URL as its content.

29.4 Defining and redefining marker behavior

In this section:

§29.4.1 [Assigning properties to marker types](#) on page 838

§29.4.2 [Observing restrictions on redefining marker behavior](#) on page 840

§29.4.3 [Understanding examples of marker redefinition](#) on page 840

29.4.1 Assigning properties to marker types

You can define the behavior of a new marker type, or redefine the behavior of an existing marker type, by assigning one or more properties to the marker type:

```
[MarkerTypes]
; marker type name = properties
FM_Marker = Property1 Property2 ...
```

*Most properties
are specific to
HTML or to RTF*

Which properties you can assign depends on whether you are converting to HTML or to RTF. A few properties are common to both output types: Delete, Code, ALink, and Config. The rest are specific to either HTML or RTF output. [Table 29-3](#) lists all the [MarkerTypes] properties, shows which output types apply, and describes the effect of each property.

*Marker types lose
original properties*

When you assign properties to a FrameMaker marker type in [MarkerTypes], that marker type loses its original FrameMaker functionality; instead, it takes on the properties you assign to it.

*Marker types
remapped in
[Markers] are
gone*

If you remap a marker type to something else in [Markers], *unless you also remap that type to itself*, **Mif2Go** ignores [MarkerTypes] property assignments to the original marker type. In fact, for conversion purposes, the original marker type is gone. However, **Mif2Go** does honor [MarkerTypes] property assignments to any *new* marker types you define in [Markers]. See §29.3 [Remapping marker types and hypertext commands](#) on page 836.

Table 29-3 Effects of [MarkerTypes] properties

Output	Property	Effect
RTF or HTML	ALink	Content is treated as a list of names of categories that apply to the current topic. Category names should be single terms, separated by semicolons. Use spaces or other punctuation in the names at your own risk. Available for WinHelp, MS HTML Help, OmniHelp, and Oracle Help for Java.
	Code	Any macros in the marker are expanded, and the content is surrounded by any code specified for the marker type in [MarkerTypeCodeBefore] and [MarkerTypeCodeAfter]; see §29.7 Inserting code or text with markers on page 842 for more information. <i>Cannot be combined with HTMLComment.</i> Compare with property Text for generic XML or HTML/XHTML output.
	Config	Content is a configuration setting of the form [Section]Key=Value or [Section]=Value; see §33.2 Overriding settings with markers or macros on page 920.
	Delete	The marker is removed entirely; Mif2Go applies this property last, after any other properties you specify. <i>Must be specified last.</i>
RTF only	RTFConfig	Content is a configuration setting of the form [Section]Key=Value or [Section]=Value; see §33.2 Overriding settings with markers or macros on page 920.
HTML only	ANSI	Specifies the Windows code page to use for FrameMaker, default 1252; or 1250 for CE/EE, 1251 for Cyrillic, 1253 for Greek, 1254 for Turkish. See §13.4.3.2 Selecting a Windows code page for single-byte character sets on page 431.
	Extr*	Each of these markers has the same effect as the corresponding [HTMLParaStyles]parafmt=Extr* property; see §18.3.1 Enabling and disabling extract processing on page 591. For ExtrDisable, ExtrEnable, ExtrEnd, and ExtrStart, any content is ignored, unless you also specify other properties that use the content, such as Code.
	ExtrBottom	Content becomes the last item in the extract <body>.
	ExtrDisable	Turns extract processing off.
	ExtrEnable	Turns extract processing on.
	ExtrEnd	Ends a file extract, but is not part of the extract.
	ExtrFinish	Ends a file extract, and is the last part of the extract.
	ExtrHead	Content is placed in the <head> of the extract, after the <title> element.
	ExtrReplace	Content replaces an extract in the parent file.
	ExtrStart	Begins an extract. <i>Must be specified before FileName or Title.</i>
	ExtrTop	Content becomes the first item in the extract <body>.
	FileName	Marker content names the current split or extracted file; <i>dangerous</i> (see §34.8.3 Using custom markers to name output files on page 947)
	HelpMerge	Marker content specifies another help file to be merged at the point of insertion
	HTMConfig	Content is a configuration setting of the form [Section]Key=Value or [Section]=Value; see §33.2 Overriding settings with markers or macros on page 920.
	HTMLComment	Marker content is treated as an HTML comment, and enclosed within HTML comment delimiters; or, if you specified XML as the output type, marker content is properly converted to an XML comment. <i>Cannot be combined with Code.</i>
	Split	Marks a split point in a FrameMaker file; has the same effect as [HTMLParaStyles]parafmt=Split; see §18.2.1 Designating split points on page 586. Any content is ignored, unless you also specify other properties that use the content, such as Code. <i>Must be specified before FileName or Title.</i>
	Text	Marker content is processed as pure text per the current text encoding rules; non-alphanumeric characters are treated as text and properly encoded for output. Applies to generic XML and HTML/XHTML output only; see §29.7.3 Processing marker content as text for XML/HTML/XHTML on page 844. Compare with property Code.

Table 29-3 Effects of [MarkerTypes] properties (continued)

Output	Property	Effect
	Title	Marker content becomes the page title attribute of the current split or extract file.
	TopicStartCode	Same as the Code property, except macros are expanded at the start of the topic. Any output the macros create is available as predefined macro <\$_TopicStartCode>, which can be used anywhere in the current topic.
	Window	<i>HTML Help only.</i> Marker content names a secondary window as the target for jumps from the paragraph containing the marker.

29.4.2 Observing restrictions on redefining marker behavior

Marker types you cannot redefine You cannot use [MarkerTypes] to redefine the properties of any of the following FrameMaker marker types:

Cross-Ref
HTML Macro
Hypertext
Index

However, you can remap any of these marker types to another marker type in [Markers] (see §29.3 [Remapping marker types and hypertext commands](#) on page 836), and then list the other type in [MarkerTypes] and redefine its properties.

Marker types you can redefine You can redefine the properties of any of the following:

- any new marker type you add in FrameMaker
- any new marker type you introduce in [Markers]
- standard FrameMaker marker types other than **Index**, **Hypertext**, **Cross-ref**, or **HTML Macro**.

Note: If you want to redefine the properties of a standard FrameMaker marker type that you remap to another type in [Markers], *you must also remap the original marker type to itself* in [Markers] before you can redefine its properties in [MarkerTypes]; see §29.3.1 [Remapping and cloning marker types](#) on page 836.

Order of properties is important

Mif2Go processes marker-type properties from left to right, and in some cases the order in which you list property names is important:

- Split or ExtrStart should be specified before (to the left of) Cross-Ref, Title, or FileName; otherwise the cross reference, page title, or file name is likely to be applied to the prior file segment.
- Cross-Ref should be specified before Delete but after all other property names.
- Delete should be specified last.

29.4.3 Understanding examples of marker redefinition

Check the following examples for ways to use redefined markers:

[Example: redefining Subject marker type](#)

[Example: extracting content from markers](#)

[Example: combining Split and FileName properties](#)

[Example: redefining HTML Macro marker type](#)

Example:
redefining Subject marker type

You could redefine the behavior of the original **Subject** marker type in the example in §29.3 [Remapping marker types and hypertext commands](#) on page 836 so that after remapping **Subject** markers, the original **Subject** markers could be used to mark split points instead. You would specify:

```
[Markers]
Subject = Index ALinkRef Subject

[MarkerTypes]
Subject = Split
```

Or, you could simply specify `Split` as a marker type on the right in `[Markers]`:

```
[Markers]
Subject = Index ALinkRef Split
```

This works because when you give a marker type the same name as a property, **Mif2Go** takes that as an implicit request to use that marker type to assign that property. In effect, each `[MarkerTypes]` property name can also be used as the name of a predefined marker type. See §29.6 [Using marker property names for marker types](#) on page 842 for more information.

Example:
*extracting content
from markers*

To roll your own macros for related-topic buttons in HTML Help, you could capture ALink keywords from **Subject** markers, remap **Subject** markers to a new marker type, and also clone the resulting markers:

```
[Markers]
Subject = AKey ALink
```

See §9.7.5.2 [Creating a list of ALink keywords from markers](#) on page 314 for a description of how these markers and their contents can be used to build button macros.

Example:
*combining Split
and FileName
properties*

As another example, you can combine `Split` and `FileName` properties in one marker type, as follows:

1. Add a new custom marker type in FrameMaker, called (for example) **SplitFile**.
2. Make the content of each **SplitFile** marker the base file name (no extension).
3. Specify the following setting for the **SplitFile** marker type:

```
[MarkerTypes]
SplitFile = Split FileName
```

Example:
*redefining HTML
Macro marker
type*

You cannot specify both `HTMLComment` and `Code` properties for the same marker type. However, you can specify both as marker types in `[Markers]`, and each will have its usual effect. You might want to do this, to include as a comment the macro that will be expanded in the HTML code. This can help when you are setting up a macro system where macros call other macros; you can see what you asked for, and what you got.

To make each FrameMaker **HTML Macro** marker into a custom **HTMLComment** marker, plus a custom **Code** marker, with macro expansion:

```
[Markers]
HTML Macro = HTMLComment Code
```

29.5 Suppressing markers

To prevent a marker type from being included in your output, you can map it out of existence by assigning property `Delete`. For example, to eliminate FrameMaker index entries from RTF output:

```
[Markers]
Index = Delete
```

See §29.4 [Defining and redefining marker behavior](#) on page 838.

29.6 Using marker property names for marker types

To simplify marker use, you can specify any [MarkerTypes] property name as the name of a marker type, and get the same effect as assigning that property to some other marker type. If you add a custom FrameMaker marker type that has the same name as one of the properties listed in [Table 29-3](#) on page 839, the marker type takes on that property.

This is how it works: **Mif2Go** looks up, in [MarkerTypes], the name of each marker type that you have either:

- used in your FrameMaker document, or
- added to the right of the = in [Markers].

If you listed the name of that marker in [MarkerTypes], **Mif2Go** processes each property you specified for it, and treats each property as though you had added a marker that has:

- the name of the property,
- the effect of the property, and
- the content of the marker.

For example, to mark split points for HTML output (see §18.2 [Splitting files](#) on page 586), you could add a custom marker type named **Split** to your FrameMaker document (or list **Split** to the right of the = in [Markers]), and insert a **Split** marker wherever you want the file split. This would have the same effect as inserting some other marker to which you assign the Split property in [MarkerTypes].

29.7 Inserting code or text with markers

You can have **Mif2Go** include the content of a marker in the output, and optionally surround it with any code you provide.

In this section:

§29.7.1 [Inserting marker content in output](#) on page 842

§29.7.2 [Surrounding marker content with code](#) on page 843

§29.7.3 [Processing marker content as text for XML/HTML/XHTML](#) on page 844

§29.7.4 [Surrounding attribute markers with code](#) on page 845

§29.7.5 [Converting custom markers to attributes](#) on page 845

§29.7.6 [Including code to be executed before a topic](#) on page 846

29.7.1 Inserting marker content in output

Mif2Go can insert the content of a marker directly in output, at the location where it occurs in your FrameMaker document. This is what happens by default to the content of any marker of type **Code** or **Text**, or of any marker that is remapped to **Code** or **Text** (see §29.3 [Remapping marker types and hypertext commands](#) on page 836).

If **Code** marker content includes **Mif2Go** macros, the macros are expanded; however, macros in **Text** markers are not expanded.

To place marker content at a location in HTML or XML that is outside of any paragraph, dedicate a paragraph format in FrameMaker to this use (for example, *MarkerOnly*). Put the marker in an otherwise empty *MarkerOnly* paragraph, and assign the following property:

```
[HTMLParaStyles]
MarkerOnly = Raw
```

See §21.3.6 [Stripping paragraph properties](#) on page 650.

See also:

§14.8.1 [Configuring index markers for conversion to XML](#) on page 469

§29.7.2 [Surrounding marker content with code](#) on page 843.

§29.7.3 [Processing marker content as text for XML/HTML/XHTML](#) on page 844.

29.7.2 Surrounding marker content with code

If you assign the `Code` property or the `Text` property to a marker type (see §29.4.1 [Assigning properties to marker types](#) on page 838), you can have **Mif2Go** surround the content with additional “before” and “after” code, or replace the content with code:

```
[MarkerTypeCodeBefore]
; marker type name = macro
; for markers assigned the Code or Text property in MarkerTypes.

[MarkerTypeCodeAfter]
; marker type name = macro
; for markers assigned the Code or Text property in MarkerTypes.

[MarkerTypeCodeReplace]
; marker type name = macro
; for markers assigned the Code or Text property in MarkerTypes.
```

Mif2Go expands macros assigned in these sections. If the marker type is assigned the `Code` property and marker content includes macros, those macros are expanded, also. If the marker type is assigned the `Text` property, marker content is treated as plain text.

ALink references

Following the example in §29.3 [Remapping marker types and hypertext commands](#) on page 836, to create an `ALink` reference to the link identified in the marker content:

```
[Markers]
Subject = Index ALinkRef

[MarkerTypes]
ALinkRef = Code

[MarkerTypeCodeBefore]
ALinkRef=<a href="alink:

[MarkerTypeCodeAfter]
ALinkRef=">Related Topics</a>
```

In this example, if you are generating Oracle Help for Java, you specify one subject name in the **ALinkRef** marker, and get both an `ALink` (which makes the current topic a member of the group) and a hotspot that calls up that list of topics.

Macro variables

You can capture marker content in a macro variable at the same time. For example:

```
[Markers]
ALink = ALink AName

[MarkerTypes]
AName = Code

[MarkerTypeCodeBefore]
AName = <$$ALinkText= "

[MarkerTypeCodeAfter]
AName = ">
```

Mif2Go first makes a copy (**AName**) of the **ALink** marker, then uses that copy to create a macro assignment statement:

```
<$$ALinkText= "marker content">
```

This statement sets the value of the macro variable to the content of the marker, without producing any output; and you can use the macro variable in other macros. Because the same variable may be assigned multiple times in a document, **Mif2Go** processes each assignment in document sequence. Therefore the assignment must precede the point of use.

For a more extensive example for HTML Help, see §9.7.5.2 [Creating a list of ALink keywords from markers](#) on page 314.

*Content
replacement*

You can replace marker content at the same time; for example, to substitute a numeric entity for a special character:

```
[MarkerTypeCodeBefore]
HeaderFooter2=<ph outputclass="HeaderFooter2"><$$string="
[MarkerTypeCodeAfter]
HeaderFooter2="><$( $$string replace "@" with "&#174;" )></ph>
```

For marker content “Vertigo®” the result would be:

```
<ph outputclass="HeaderFooter2">Vertigo&#174;</ph>
```

See §28.6.5 [Specifying substrings in expressions](#) on page 817.

29.7.3 Processing marker content as text for XML/HTML/XHTML

For most output types, **Mif2Go** processes FrameMaker **Index** markers for their usual purpose. However, for generic XML or HTML/XHTML output, **Mif2Go** makes no assumptions about the meaning of **Index** markers. If you wish to have the content of FrameMaker **Index** markers included in output as, for example, <indexterm> elements, you must direct **Mif2Go** to surround the marker content with appropriate tags; see §29.7.2 [Surrounding marker content with code](#) on page 843.

For example:

```
[MarkerTypeCodeBefore]
Index = <indexterm>
[MarkerTypeCodeAfter]
Index = </indexterm>
```

*Special
characters in
marker content*

Suppose some of your FrameMaker **Index** markers happen to include text surrounded by angle brackets, such as this example:

```
\<a name=...> tags
```

For generic XML output, with the above code-before and code-after settings, **Mif2Go** would render this marker content as:

```
<indexterm><a name=...> tags</indexterm>
```

This is not valid XML; what you really want in generic XML output is this:

```
<indexterm>&lt;a name=...&gt; tags</indexterm>
```

To achieve the correct encoding of the angle brackets you must create a new marker type, cloning (and replacing) existing markers of type **Index**, and assigning the Text property to the new marker type. For example:

```
[Markers]
; Clone and replace markers of type Index:
Index = NewIndex
```

(When you specify Index=NewIndex, rather than Index=Index NewIndex, the original **Index** markers are no longer in effect for conversion purposes.)

```
[MarkerTypes]
; Assign the Text property to markers of type NewIndex:
NewIndex = Text

[MarkerTypeCodeBefore]
; Precede the content of each NewIndex marker with an opening tag:
NewIndex = <indexterm>

[MarkerTypeCodeAfter]
; Follow the content of each NewIndex marker with a closing tag:
NewIndex = </indexterm>
```

Because you have assigned the Text property to markers of type **NewIndex**, **Mif2Go** treats the marker content as plain text, and applies the appropriate encoding to non-alphanumeric characters.

Note: You do not need this approach for DITA or DocBook XML output, nor for any of the Help output types, nor for RTF output.

See also:

§14.8.1 [Configuring index markers for conversion to XML](#) on page 469

29.7.4 Surrounding attribute markers with code

You can specify `[MarkerTypeCodeBefore]` and `[MarkerTypeCodeAfter]` for attribute markers such as **LinkTitle**. However, for attribute markers you do *not* assign the Code property to the attribute marker type in `[MarkerTypes]`, because that would insert marker code on the spot *instead of* in an attribute marker.

For example, you could use the same content for the link title and `LinkonMouseOver` attributes, and provide wrapper code for the latter, with just a single **LinkTitle** marker in FrameMaker:

```
[Markers]
LinkTitle=LinkTitle LinkonMouseOver

[MarkerTypeCodeBefore]
LinkonMouseOver=window.status='

[MarkerTypeCodeAfter]
LinkonMouseOver='; return true;
```

29.7.5 Converting custom markers to attributes

Suppose your FrameMaker document consists of a catalog with tables of items, including descriptions and prices. Each price cell has a custom marker, **Price**; the content of each **Price** marker is the part number of the item. And suppose in DITA XML output you want each price cell to include the text of the price, and also a `conref` attribute with the part number as the attribute value. In other words, for each cell containing a price `$xxx.xx` and a **Price** marker with content `nnnnn`, the corresponding DITA table cell should look like the following:

```
<entry><p>
  <ph conref="nnnnnn">$xxx.xx</ph>
</p></entry>
```

First you would assign the Code property to the **Price** markers (see [Table 29-3](#)):

```
[MarkerTypes]
Price = Code
```

Now you can surround the part number with the `conref` assignment:

```
[MarkerTypeCodeBefore]
Price = <ph conref="

[MarkerTypeCodeAfter]
Price = ">
```

To wrap the price text in the same <ph> element, how you proceed depends on whether the price values in the table have a unique paragraph format.

If the price text has a unique paragraph format (for example, *CurrPrice*):

```
[HTMLParaStyles]
CurrPrice = CodeEnd

[ParaStyleCodeEnd]
CurrPrice = </ph>
```

If the price text does *not* have a unique paragraph format, you would have to use a macro and a macro variable to identify cells that contain price values. For example, if price values use a general-purpose paragraph format named *TableCell*:

```
[HTMLParaStyles]
TableCell = CodeEnd

[ParaStyleCodeEnd]
TableCell = <$_if ($$isprice)></ph><$$isprice=0><$_endif>
```

Instead of just closing the conref assignment, you would also set the value of the macro variable to indicate this is a price cell:

```
[MarkerTypeCodeAfter]
Price = "><$$isprice=1>
```

You should initialize the macro variable:

```
[MacroVariables]
isprice=0
```

See §28 [Working with macros](#) on page 787.

29.7.6 Including code to be executed before a topic

Suppose you need to provide code that must be processed at the start of a topic, before anything has been written to the topic file; for example, variable content to be included in the <head> element of each HTML topic.

Because the same variable can be assigned a different value multiple times in a document, **Mif2Go** processes each assignment as it is encountered. Therefore, the assignment of a particular value to a variable must ordinarily precede the point where the variable is used. To get around this restriction, you can assign property **TopicStartCode** to a marker, to have the code executed before the topic starts.

For example, to provide a Help keyword in an XML section of the <head> element, you could assign property **TopicStartCode** to a custom marker:

```
[MarkerTypes]
FlKeyword = TopicStartCode

[MarkerTypeCodeBefore]
FlKeyword = <$$FlKey = "

[MarkerTypeCodeAfter]
FlKeyword = ">

[Inserts]
Head = <$KeyIndexF>
```

In macro <\$KeyIndexF>, you would include code such as the following:

```
<Help:Keyword Index="F" Term="<$$FlKey>" />
```

As another example, to include a variable number of keywords, each in its own <meta> tag:

```
[MarkerTypes]
MetaKeys = TopicStartCode

[MarkerTypeCodeBefore]
MetaKeys = <$$KeyCount++><$$Keywords[ $$KeyCount ] = "

[MarkerTypeCodeAfter]
MetaKeys= ">

[Inserts]
Head = <$AddKeywords>

[AddKeywords]
<$_repeat ( $$KeyCount )>
<meta name="Help.Keywords" content="<$$Keywords[ $$_count ]>" />\n
<$_endrepeat><$$KeyCount=0>

[Macros]
OmitMacroReturns=Yes

[MacroVariables]
KeyCount=0
```

In this variation, you would use a counter (\$\$KeyCount) that starts at zero. As **Mif2Go** processes the **MetaKeys** markers for the start of the topic, the counter is incremented before each marker. The counter is used to index an array variable, into which the marker content is stored. Each succeeding **MetaKeys** marker gets its own slot in the array.

When **Mif2Go** is ready to write the <head> of the topic, the <\$_repeat> loop (see §28.6.4.3 [Using loop structures](#) on page 816) writes as many <meta> tags as there were **MetaKeys** markers, each with the content from one marker, and the counter is set back to zero for the next file.

29.8 Identifying markers with variable <\$\$_objectid>

Suppose your workflow requires distinguishing between two or more markers (index entries, for example) that have the same content. **Mif2Go** assigns a unique ID to each marker, of the form Xaa123456, composed as follows:

X	Fixed character
aa	Mif2Go FileID for the file containing the marker
123456	FrameMaker ObjectID for the marker

Mif2Go makes this marker ID available as predefined macro variable <\$\$_objectid>. So you could do something like this:

```
[Markers]
Index=MyIndex

[MarkerTypes]
MyIndex=Code

[MarkerTypeCodeBefore]
MyIndex= <a name="<$$_objectid>" class="Myindex" value="

[MarkerTypeCodeAfter]
MyIndex= "></a>
```


30 Working with templates

This section explains how to use **Mif2Go** configuration templates and how to import FrameMaker templates. Topics include:

- §30.1 [Working with configuration templates](#) on page 849
- §30.2 [Referencing configuration files and templates](#) on page 851
- §30.3 [Including document-specific configuration files](#) on page 852
- §30.4 [Including chapter-specific configuration files](#) on page 855
- §30.5 [Deciding which configuration file to edit](#) on page 856
- §30.6 [Creating your own configuration templates](#) on page 861
- §30.7 [Applying FrameMaker conversion templates](#) on page 863

30.1 Working with configuration templates

A *configuration template* is a configuration file that contains settings that can be referenced by (and thus be included in) another configuration file. **Mif2Go** relies heavily on configuration templates to supply settings that seldom vary from project to project, or from one output type to another. This approach helps eliminate duplication of settings, and reduces the clutter in your project configuration file by limiting the latter to just those settings specific to your current project.

In this section:

- §30.1.1 [Understanding how templates are organized](#) on page 849
- §30.1.2 [Understanding how templates are named](#) on page 850
- §30.1.3 [Understanding how templates are chained together](#) on page 850
- §30.1.4 [Understanding how macro libraries are organized](#) on page 851

30.1.1 Understanding how templates are organized

Your **Mif2Go** distribution includes the collections of configuration templates listed in [Table B-1](#) on page 1017. These templates are linked together by references that extend through the chains, from your project configuration file through templates in %OMSYSHOME%\m2g subdirectories, to the very end of the configuration chain at %OMSYSHOME%\common\system\config\omsys.ini. See §30.1.3 [Understanding how templates are chained together](#) on page 850

Configuration templates include the following groups:

- [General configuration settings](#), located in config directories
- [Macro definitions](#), located in macros directories

Each template in a system subdirectory is paired with an editable configuration file in the corresponding local subdirectory. The system member of each pair contains default settings. The local member of each pair starts out empty; you add settings to override the default settings referenced in the system member. Your configuration files always reference the local member. The local member references its system counterpart, which in turn references the local member of the next template up the chain.

*General
configuration
settings*

The settings in general configuration templates establish default values for features that are not likely to differ from one project to the next, or from one output type to the next. General configuration templates are located in the following directories:

%OMSYSHOME%\m2g\system\config: *default general configuration settings*

%OMSYSHOME%\m2g\local\config: *your overrides to the system settings*

Macro definitions

General-purpose macros are defined in macro library templates. Each configuration section in these templates is the name of the macro being defined. Macro library templates are located in the following directories:

%OMSYSHOME%\m2g\system\macros: *definitions of general-purpose macros*

%OMSYSHOME%\m2g\local\macros: *definitions of your own macros*

30.1.2 Understanding how templates are named

Template files in your **Mif2Go** distribution have names that follow a certain pattern. Each name has a prefix that indicates the scope of the settings the template contains, followed by the type of template (config or macro):

<u>Template name</u>	<u>Editable file name</u>	<u>Scope of settings</u>
omsys.ini	local_omsys.ini	All Omni Systems projects
m2g_type.ini	local_m2g_type.ini	All Mif2Go projects
m2htm_type.ini	local_m2htm_type.ini	Mif2Go HTML and XML projects
m2rtf_type.ini	local_m2rtf_type.ini	Mif2Go RTF projects

Most general configuration templates and macro libraries use this naming convention.

30.1.3 Understanding how templates are chained together

Configuration templates are chained together in a series, each accessing all the settings in the next, plus all the settings in all other templates farther up the chain. If a given setting appears in more than one template in a chain, the instance of that setting closest to your project configuration file takes precedence over any that are farther away.

Table 30-1 Output-type-specific general configuration files

Output type	Project configuration file	Editable local configuration file
DITA	_m2dita.ini	local_m2dita_config.ini
DocBook	_m2docbook.ini	local_m2docbook_config.ini
Eclipse Help	_m2eclipse.ini	local_m2eclipse_config.ini
HTML	_m2html.ini	local_m2html_config.ini
MS HTML Help	_m2htmlhelp.ini	local_m2htmlhelp_config.ini
JavaHelp	_m2javahelp.ini	local_m2javahelp_config.ini
OmniHelp	_m2omnihelp.ini	local_m2omnihelp_config.ini
Oracle Help	_m2oraclehelp.ini	local_m2oraclehelp_config.ini
WinHelp	_m2winhelp.ini	local_m2winhelp_config.ini
MS Word	_m2rtf.ini	local_m2rtf_config.ini
XHTML	_m2xhtml.ini	local_m2xhtml_config.ini

Each project configuration file references an output-type-specific local configuration file. This editable local configuration file in turn references its system counterpart, which references the next editable local configuration file in the chain, and so forth.

For example, the MS Word starting project configuration file `_m2rtf.ini` references this chain of general configuration templates and files:

```
_m2rtf.ini ->
  local_m2rtf_config.ini-> m2rtf_config.ini ->
    local_m2g_config.ini -> m2g_config.ini ->
      local_omsys.ini -> omsys.ini
```

The HTML starting project configuration file references this chain:

```
_m2html.ini ->
  local_m2htm_config.ini -> m2htm_config.ini ->
    local_m2g_config.ini -> m2g_config.ini ->
      local_omsys.ini -> omsys.ini
```

Some chains are longer. For example, for OmniHelp output, the chain looks like this:

```
_m2omnihelp.ini ->
  local_m2omnihelp_config.ini -> m2omnihelp_config.ini ->
    local_m2help_config.ini -> m2help_config.ini ->
      local_m2htm_config.ini -> m2htm_config.ini ->
        local_m2g_config.ini -> m2g_config.ini ->
          local_omsys.ini -> omsys.ini
```

All general configuration chains go through either `m2htm_config.ini` (for HTML or XML output) or `m2rtf_config.ini` (for Word or WinHelp output). These two configuration templates reference the macro configuration files and templates, through side chains. Therefore, as long as your project configuration file references one of the output-specific configuration files, you do not have to include settings in your project configuration file to reference those other files.

See also:

§30.1.4 [Understanding how macro libraries are organized](#) on page 851

30.1.4 Understanding how macro libraries are organized

Two **Mif2Go** general configuration templates, `m2htm_config.ini` and `m2rtf_config.ini`, respectively reference `local_m2htm_macros.ini` and `local_m2rtf_macros.ini`; and these two editable macro library files in turn reference `m2htm_macros.ini` and `m2rtf_macros.ini`, respectively.

For example, in `m2htm_config.ini`:

```
[Templates]
; Macros = path to macro library
Macros = %OMSYSHOME%\m2g\local\macros\local_m2htm_macros.ini
```

Therefore you do not have to include a setting for `Macros` in your starting project configuration file, unless you wish to include an additional macro library in the chain. See §28.2.4 [Including macro definitions in your own macro library](#) on page 794.

30.2 Referencing configuration files and templates

To reference a **Mif2Go**-provided general configuration file or template:

```
[Templates]
Configs = %OMSYSHOME%\path\to\sometemplate.ini
```

This setting takes the place of the deprecated `[FDK]ConfigTemplate` setting. Replace the old `[FDK]` setting with the new `[Templates]` setting in all your configuration files. Your **Mif2Go** distribution includes configuration templates already chained together through references like this; see §30.1.3 [Understanding how templates are chained together](#) on page 850.

When **Mif2Go** creates a starting configuration file for a new project, that file includes the first link in the chain. For example, a starting project configuration file for Word output includes this reference:

```
[Templates]
Configs = %OMSYSHOME%\m2g\local\config\local_m2rtf_config.ini
```

If you want to insert another configuration file (for example, `myspecial.ini`) in the chain between the project configuration file and `local_m2rtf_config.ini`, you would copy this reference into `myspecial.ini`, and replace it with the following reference in the project configuration file:

```
[Templates]
Configs = relative\path\to\myspecial.ini
```

Make paths to your own configuration files relative to the referencing configuration file. You can chain configuration files together by including in each a `[Templates]Configs` setting that references yet another template or configuration file. You can have as many referenced files chained as you please; each overrides the one it references, and all others that precede the referenced template in the chain. The most specific configuration rules. See §30.6 [Creating your own configuration templates](#) on page 861.

Settings that specify paths to configuration templates, or to any other files in the **Mif2Go** distribution directory structure, should use absolute paths that begin with environment variable `%OMSYSHOME%`; for example:

```
Configs = %omsyshome%\m2g\local\config\local_m2htm_config.ini
```

If you specify a relative path for any setting in configuration section `[Templates]`, that path is considered to be relative to the configuration file in which the setting occurs.

Precedence of settings

If the same setting has different values in a referenced template or configuration file and in a file that references that template, the value in the referencing file takes precedence, allowing you to override the template when necessary:

- The last referenced file in the chain overrides **Mif2Go** internal default values.
- A referenced configuration file overrides, in turn, any files it references.
- A document-specific configuration file overrides any files it references, and also overrides any other files the starting project configuration file references
- The starting project configuration file overrides any files it references.
- Any chapter-specific configuration file overrides the project configuration file, for its FrameMaker chapter only.

See §33.1.2 [Understanding precedence of configuration settings](#) on page 919.

30.3 Including document-specific configuration files

In addition to a general chain of configuration files and templates, your project configuration file references a chain of configuration files containing settings that apply only to the current FrameMaker document (though possibly to multiple output types from that document).

In this section:

- §30.3.1 [Understanding document-specific configuration files](#) on page 853
- §30.3.2 [Referencing a document-specific configuration file](#) on page 853
- §30.3.3 [Deciding where to keep document-specific configuration files](#) on page 854
- §30.3.4 [Indicating the intended scope of a configuration file](#) on page 855

30.3.1 Understanding document-specific configuration files

When you set up a conversion project, **Mif2Go** creates a document-specific configuration file for you, and places a reference to that file in your project configuration file; see §3.5 [Understanding how Mif2Go sets up a project](#) on page 82. To establish a document-specific configuration, **Mif2Go** looks in this file:

```
%omsyshome%\m2g\local\config\local_m2g_config.ini
```

or, if the required settings are not there, in this file:

```
%omsyshome%\m2g\system\config\m2g_config.ini
```

for the following settings:

```
[Setup]
. . .
; used when creating document-specific configuration files
LocalConfigPath = ..\_config\
WinHelpDocName = winhelp_doc.ini
WordDocName = word_doc.ini
HTMLDocName = html_doc.ini
```

These are default values; see §1.3.6 [Establish system-wide configuration settings](#) on page 58.

If the directory named by `LocalConfigPath` is not already present, **Mif2Go** creates this directory.

If not already present in the directory named by `LocalConfigPath`, **Mif2Go** creates a configuration file there with one of the following names (or other names if you have changed the defaults), depending on the output type of the project:

```
html_doc.ini      for HTML, HTML-based Help, or XML projects
word_doc.ini      for Word or WordPerfect projects
winhelp_doc.ini   for WinHelp projects
```

This document-specific configuration file is intended for settings that are likely to apply only to the current FrameMaker document, and that will be the same for most outputs. Initially this file includes settings in the following sections:

```
html_doc.ini      [HTMLParaStyles], [HelpContentsLevels]
word_doc.ini      [WordSectionFiles], [WordCntStyles]
winhelp_doc.ini   [HelpStyles], [HelpCntStyles],
                  [BrowseStart]
```

Also included is a comment to show that this file is referenced from the project configuration file. For example:

```
; Document-specific configuration for HTML outputs
; Referenced by _m2htm.ini
```

If a document-specific configuration file for the current output type is already present in the directory named by `LocalConfigPath`, **Mif2Go** does not overwrite that file, nor add a comment. For additional conversions from the same document you might want to add “referenced by” comments yourself, so you can keep track of which of your project configuration files reference the document-specific configuration file.

30.3.2 Referencing a document-specific configuration file

To reference a document-specific configuration file:

```
[Templates]
; Document = path to document-specific configuration file
Document = %OMSYSHOME%\m2g\documents\mydocname_doc.ini
```

Or:

```
[Templates]
; Document = path to local document-specific configuration file
Document = mydocsource\_config\outputtype_doc.ini
```

It is a matter of preference whether you keep document-specific configuration files for all documents in one central location (for example, the %OMSYSHOME%\m2g\documents directory included in your **Mif2Go** distribution for this purpose) or in a `_config` subdirectory under your FrameMaker source directory; see §30.3.3 [Deciding where to keep document-specific configuration files](#) on page 854.

Mif2Go processes the entire `Document` chain before continuing with the `Configs` chain. The `Document` chain is interpolated between your starting project configuration file and the `Configs` chain that your project configuration file references. Settings in the `Document` chain override settings in the `Configs` chain.

Although it cannot reference general configuration files, a document-specific configuration file can reference other types of configuration files via the following `[Templates]` settings:

<code>Document</code>	Other document-specific configurations
<code>Macros</code>	Macro libraries

Settings in files referenced by a document-specific configuration file override settings in other types of files referenced by your project configuration file. Settings in your project configuration file override settings in the `Document` chain.

30.3.3 Deciding where to keep document-specific configuration files

You can establish a default location for configuration files that contain settings that apply to all conversion projects for a particular FrameMaker document; for example, settings that name the document itself, or that reference values that occur only in that document. You can use a different location for each source document, and you can change the location for each project. Choose a default that applies to most of your projects, depending on which of several possible scenarios is most likely:

- [Single output type per document](#)
- [Multiple output types per document](#)
- [Multiple documents per output type](#)
- [Configurations shared on a network](#)
- [Lone writer](#)
- [Need for portability.](#)

*Single output type
per document*

If you expect to produce only one output type from each FrameMaker document, and no one else will be working on the same document, you really do not need a document-specific configuration file at all; your project configuration file can include all the needed settings. The best place is a location relative to your FrameMaker document files; the default location is a directory named `_config`, parallel to your project directory.

*Multiple output
types per
document*

If you expect to produce more than one output type from each FrameMaker document, choose a location relative to your FrameMaker document files; the default location is a directory named `_config`, parallel to your project directory.

<i>Multiple documents per output type</i>	If you expect to produce the same output type from multiple source documents, you might want to keep source-specific configuration files for all documents in the same directory tree, for easy comparison; this would be a reason to choose a location central to all your Mif2Go projects such as the directory provided in your distribution, %OMSYSHOME%\m2g\document.
<i>Configurations shared on a network</i>	If %OMSYSHOME% is located on a network drive (<i>not advisable</i>), and more than one person will need to access document-specific settings for the same document, you might want to choose a location central to all your Mif2Go projects.
<i>Lone writer</i>	If you are the only person working on your projects, a location central to all your Mif2Go projects is probably easier: you have all the document-specific configuration files in one area, and if you want to see how you did something in another project, you can easily find the configuration file for that project.
<i>Need for portability</i>	If you need to be able to move entire projects from one machine to another, or if you might have to pack up a project and send it to someone else, choose a location relative to your FrameMaker document files.

30.3.4 Indicating the intended scope of a configuration file

To show the intended scope of settings in a configuration file, you can include the following setting:

```
[Templates]
; Scope = Intended scope, such as "All Word guides for Product A"
Scope = Statement of intended scope
```

The value of *Scope* is displayed by the Configuration Manager. If you add a *Scope* setting to a configuration file included in your **Mif2Go** distribution, that value overrides any internal *Scope* value maintained by the Configuration Manager.

30.4 Including chapter-specific configuration files

A chapter-specific configuration file contains settings that apply only to a single chapter file in your FrameMaker document. Each chapter-specific configuration file is named for its FrameMaker chapter file, with extension *.ini*.

Chapter-specific configuration files do not include a `[Templates]Configs` setting; **Mif2Go** inserts each at the very bottom of the configuration chain, below the project configuration file, and processes the chapter configuration first. A chapter configuration file can override anything in the project configuration file *except* the `[Templates]Configs` setting.

[Table 30-2](#) shows the chain of configuration files and templates for the title-page file of the **Mif2Go User's Guide**, for the HTML edition.

Table 30-2 Configuration chain for Mif2Go User's Guide title page

Configuration file	Contains settings for:	References:	Via:
ugmif2go.ini	User's Guide title-page chapter	_m2html.ini	(Internal reference)
_m2html.ini	User's Guide Standard HTML output project	m2gug_htm_document.ini local_m2htm_config.ini	Document Configs
m2gug_htm_document.ini	All User's Guide HTML/XML projects	m2gug_document.ini	Document
m2gug_document.ini	All User's Guide projects	(None)	
local_m2htm_config.ini	All HTML/XML projects at this site	m2htm_config.ini	Configs

Table 30-2 Configuration chain for Mif2Go User's Guide title page

Configuration file	Contains settings for:	References:	Via:
m2htm_config.ini	System defaults for all Mif2Go HTML/XML projects	local_m2g_config.ini	Configs
local_m2g_config.ini	All Mif2Go projects at this site	m2g_config.ini	Configs
m2g_config.ini	System defaults for all Mif2Go projects	local_omsys.ini	Configs
local_omsys.ini	All Omni Systems projects at this site	omsys.ini	Configs
omsys.ini	System defaults for all Omni Systems projects	(None)	

The title-page chapter has its own chapter configuration file (`ugmif2go.ini`) with settings specific to that chapter alone; those settings override any values for the same settings in the project configuration file (`_m2html.ini`).

The project configuration file (`_m2html.ini`) for the HTML edition of the **Mif2Go User's Guide** references (and overrides settings in) a configuration template named `m2gug_htm_document.ini` that contains settings common to all HTML and XML output types for the **Mif2Go User's Guide**.

In turn, `m2gug_htm_document.ini` references (and overrides settings in) `m2gug_document.ini`, which contains settings common to all **Mif2Go User's Guide** projects, including those for RTF output types.

The project configuration file also references the general chain of configuration files, including all local and system configurations, from `local_m2htm_config.ini` through `omsys.ini`.

See also:

§4.1 [Working with Mif2Go configuration files](#) on page 91

§33.1 [Using a different configuration for selected files](#) on page 919

30.5 Deciding which configuration file to edit

Each **Mif2Go** project includes one or more chains of configuration files and templates, all ultimately referenced from the project configuration file in the project directory. Which file you work with depends on the type and scope of settings you wish to add, delete, or modify.

In this section:

§30.5.1 [Understanding what configuration files are available](#) on page 857

§30.5.2 [Editing a project configuration file](#) on page 858

§30.5.3 [Editing a document-specific configuration file](#) on page 859

§30.5.4 [Editing an output-specific configuration file](#) on page 860

§30.5.5 [Editing a macro configuration file](#) on page 861

§30.5.6 [Indicating the intended scope of a configuration file](#) on page 861

See also:

§1.3.6 [Establish system-wide configuration settings](#) on page 58

§30.4 [Including chapter-specific configuration files](#) on page 855

§33.1 [Using a different configuration for selected files](#) on page 919

30.5.1 Understanding what configuration files are available

Most of the supplied **Mif2Go** configuration files available for editing are located in subdirectories of %OMSYSHOME%\m2g\local, with the following exceptions:

- Starting project configuration file, copied by **Mif2Go** from %OMSYSHOME%\m2g\local\starts (or from %OMSYSHOME%\m2g\system\starts) to your project directory
- Document-specific configuration files you have placed in the **Mif2Go** documents subdirectory or in the document _config subdirectory; see §30.3.3 [Deciding where to keep document-specific configuration files](#) on page 854
- Site-specific configuration file local_omsys.ini, located in %OMSYSHOME%\common\local\config.

[Table 30-3](#) lists the types of configuration files, shows where the files are located, and indicates the intended scope of settings for each type.

Table 30-3 Intended scope of settings by configuration type

Type	Orientation	Location of file(s)	Intended scope of settings
General	Project-specific	Project directory	Current project only
	Source-specific	.._config (parallel to project directory)	All or most outputs to be generated from a given FrameMaker source document
	Output-specific	m2g\local\config	All or most FrameMaker source documents to be converted to a given output type
	Mif2Go -wide	m2g\local\config	All Mif2Go projects for all documents and outputs
	Site-wide	common\local\config	All Omni Systems projects
Macro	Usually output-specific	m2g\local\macros	All or most outputs of a given output type or set of output types

Each configuration file in a \m2g\local subdirectory references an eponymous configuration template in a \m2g\system subdirectory that contains default settings. *Do not edit those referenced templates*, because they will be overwritten whenever you update **Mif2Go**. Instead, override settings in the corresponding \local configuration file.

[Table 30-4](#) shows a sample hierarchy of configuration files for an HTML Help project, with the most widely applicable configuration at the top of the chain, and the most narrowly applicable (the project configuration file) at the bottom. With the exception of the document-specific configuration file (shown in green), each file in [Table 30-4](#) references the file above it. The project configuration file at the bottom references both the document-specific file and the next configuration file above that.

Each configuration file in the chain can override settings in all those above it in [Table 30-4](#). This is true even for a document-specific configuration file that does not reference any of the other configuration files, because it is treated as though it were right above the project configuration file. If a document-specific configuration file does reference other configuration files, **Mif2Go** treats all their settings as overruling any files above the project configuration file in the main chain.

Note: Edit only the files shaded in blue or green; the others are system files.

Table 30-4 Chain of general configuration files for HTML Help output

Scope of settings		General configuration file
All Omni Systems projects	<i>System:</i>	%OMSYSHOME%\common\system\config\omsys.ini
	<i>Local site:</i>	%OMSYSHOME%\common\local\config\local_omsys.ini
All Mif2Go projects	<i>System:</i>	%OMSYSHOME%\m2g\system\config\m2g_config.ini
	<i>Local site:</i>	%OMSYSHOME%\m2g\local\config\local_m2g_config.ini
All HTML/XML projects	<i>System:</i>	%OMSYSHOME%\m2g\system\config\m2htm_config.ini
	<i>Local site:</i>	%OMSYSHOME%\m2g\local\config\local_m2htm_config.ini
All Help projects	<i>System:</i>	%OMSYSHOME%\m2g\system\config\m2help_config.ini
	<i>Local site:</i>	%OMSYSHOME%\m2g\local\config\local_m2help_config.ini
All HTML Help projects	<i>System:</i>	%OMSYSHOME%\m2g\system\config\m2htmlhelp_config.ini
	<i>Local site:</i>	%OMSYSHOME%\m2g\local\config\local_m2htmlhelp_config.ini
All HTML output types from this source document		.._config\html_doc.ini (parallel to project directory)
This project only		_m2htmlhelp.ini (in project directory)

*General
configuration
settings*

If you have just one FrameMaker document to convert to a single output type, most general configuration settings can go in the project configuration file; see §30.5.2 [Editing a project configuration file](#) on page 858.

If you think you might want to produce other types of output from the same source document, settings that are the same for all output types (but that would be different for other source documents) can go in a document-specific configuration file; that way you avoid duplicating the settings in every project configuration file. See §30.5.3 [Editing a document-specific configuration file](#) on page 859.

If you have many FrameMaker documents to convert to a single output type, settings that are specific to that output type and the same for every document (but that would be different for other output types) can go in the appropriate output-specific configuration file; see §30.5.4 [Editing an output-specific configuration file](#) on page 860.

*Macro
configuration files*

You can add macros to editable source-specific, output-specific, or project configuration files, as needed. Or you can add them to a macro library configuration file in m2g\local\macros. See §30.5.5 [Editing a macro configuration file](#) on page 861.

30.5.2 Editing a project configuration file

Mif2Go maintains a set of annotated templates for starting project configuration files, in directory %OMSYSHOME%\m2g\system\starts, to accommodate settings intended to apply only or primarily to individual conversion projects. *Do not modify these templates;* they will be overwritten each time you update **Mif2Go**. Instead, if you want to customize a starting project configuration file for your particular operating environment, copy the appropriate file to %OMSYSHOME%\m2g\local\starts and edit it there.

To modify settings for an individual conversion project, edit the project configuration file located in the project directory.

When you start a conversion project from within FrameMaker, **Mif2Go** copies the correct starting project configuration file for you, if one is not already present in the project directory. See §3 [Converting a book or document](#) on page 77. This new file is copied from:

- `local\starts` if a configuration file with the correct name is present; any files here are starting project configuration files you have customized to suit your operating environment.
- `system\starts` if there is no file with the correct name in the `local\starts` directory.

The output type you want to produce determines which starting project configuration file the **Mif2Go** plug-in copies to the project directory. [Table 30-5](#) lists the names of these files.

Table 30-5 Output types and starting project configuration files

Category	Output type	Project configuration file	Ref.
HTML-based Help	Eclipse Help	_m2eclipse.ini	12
	Microsoft HTML Help	_m2htmlhelp.ini	9
	JavaHelp	_m2javahelp.ini	11
	OmniHelp	_m2omnihelp.ini	10
	Oracle Help for Java	_m2oraclehelp.ini	11
HTML	Standard HTML 4.0	_m2html.ini	13
	XHTML 1.0	_m2xhtml.ini	13
XML	DITA XML	_m2dita.ini	15
	Docbook XML	_m2docbook.ini	17
	Generic XML	_m2xml.ini	14
RTF	WinHelp	_m2winhelp.ini	8
	Print RTF	_m2rtf.ini	6
Intermediate	ASCII DCL	_m2dcl.ini	38
	FrameMaker MIF	_m2mif.ini	38

Near the top of each starting project configuration file you will find a setting that links that file to an output-specific configuration file. For example, for Word output:

```
[Templates]
; Where the rest of the configuration settings are:
Configs = %omsyshome%\m2g\local\config\local_m2rtf_config.ini
```

Avoid disturbing this setting, or you might break the chain that leads to all the other configuration settings that apply to your conversion project. If you are using a source-specific configuration file, your starting project configuration file will also include a setting that links to that file; for example:

```
[Templates]
Document = g:\omnisys\ug\_config\m2gug_htm_document.ini
```

See [§30.5.3 Editing a document-specific configuration file](#) on page 859 and [§30.5.4 Editing an output-specific configuration file](#) on page 860.

30.5.3 Editing a document-specific configuration file

For settings that are specific to your FrameMaker source document and that apply to a group of output types, **Mif2Go** maintains one or more document-specific configuration files. By default, document-specific configuration files are located in a directory named `_config`, parallel to your project directory. However, you can choose to keep such files elsewhere; see [§30.3.3 Deciding where to keep document-specific configuration files](#) on page 854.

When you set up a new project, **Mif2Go** places in your project configuration file a reference to the appropriate document-specific configuration file:

```
[Templates]
Document = Path\to\mysourcedir\_config\mydocname_htm_document.ini
```

The starting project configuration file for each project references the document-specific configuration file for the document you are converting. The document-specific configuration file typically does not reference any other configuration files. However, a document-specific configuration file can reference other document-specific files (via [Templates]Document) and macro libraries (via [Templates]Macros). But a document-specific configuration file cannot reference general configuration files. See §30.3 [Including document-specific configuration files](#) on page 852.

30.5.4 Editing an output-specific configuration file

For settings that are specific to an output type or a group of output types, but that apply to all or most of your FrameMaker source documents, **Mif2Go** maintains user-modifiable output-specific general configuration files in directory %OMSYSHOME%\m2g\local\config. You can customize these files with settings that are appropriate for your particular environment. [Table 30-6](#) lists the output-specific general configuration files you can edit.

Table 30-6 Editable local output-specific configuration files

Category	Output type	Editable configuration file	Ref.
HTML-based Help	<i>All HTML-based Help outputs</i>	local_m2help_config.ini	7
	Eclipse Help	local_m2eclipse31_config.ini local_m2eclipse33_config.ini	12
	Microsoft HTML Help	local_m2htmlhelp_config.ini	9
	JavaHelp	local_m2javahelp_config.ini	11
	OmniHelp	local_m2omnihelp_config.ini	10
	Oracle Help for Java	local_m2oraclehelp_config.ini	11
HTML	<i>All HTML-based outputs</i>	local_m2html_config.ini	13
	XHTML 1.0	local_m2xhtml_config.ini	13
XML	DITA XML	local_m2dita_config.ini	15
	Docbook XML	local_m2docbook_config.ini	17
	Generic XML	local_m2xml_config.ini	14
RTF	WinHelp	local_m2winhelp_config.ini	8
	<i>All RTF-based outputs</i>	local_m2rtf_config.ini	6
Intermediate	ASCII DCL	local_m2dcl_config.ini	38
	FrameMaker MIF	local_m2mif_config.ini	38

Near the top of each configuration file you will find a setting that links that file to an eponymous configuration template in \m2g\system\config that contains settings commonly needed for the output type you selected. *Do not edit the referenced templates*; they will be overwritten each time you update **Mif2Go**. Instead, override settings as needed in the editable configuration files.

For example, in local_m2xhtml_config.ini:

```
[Templates]
; Where the rest of the configuration settings are:
Configs=%omsyshome%\m2g\system\config\m2xhtml_config.ini
```

Avoid disturbing this setting, or you might break the chain that leads to all the other configuration settings for your conversion project.

30.5.5 Editing a macro configuration file

Mif2Go provides several macro libraries in the form of macro configuration files, organized by output type. These files are located in %OMSYSHOME%\m2g\local\macros. You can add your own macros, and override macros in the default macro libraries they reference. [Table 30-7](#) lists the macro configuration files you can edit.

Table 30-7 Macro configuration files

Output type	Editable macro configuration file
HTML	local_m2g_m2htm_macros.ini
Print RTF	local_m2g_m2rtf_macros.ini
WinHelp	local_m2g_m2winhelp_macros.ini

Macro libraries are referenced from output-specific configuration files; they can also be referenced from document-specific configuration files. Each editable macro library in turn references an eponymous macro library that contains all the macros distributed with **Mif2Go**. *Do not edit those referenced libraries*; they will be overwritten every time you update **Mif2Go**. To change a distributed macro, override it with a new definition in the appropriate \local\macros library.

30.5.6 Indicating the intended scope of a configuration file

To show the intended scope of settings in a configuration file, you can include the following setting:

```
[Templates]
; Scope = Intended scope, such as "All Word guides for Product A"
Scope = Statement of intended scope
```

The value of *Scope* is displayed by the Configuration Manager. If you add a *Scope* setting to a configuration file included in your **Mif2Go** distribution, that value overrides any internal *Scope* value maintained for that file by the Configuration Manager.

30.6 Creating your own configuration templates

Besides using the configuration templates supplied with your **Mif2Go** distribution, you can create templates of your own to insert additional or alternate settings anywhere in the chains of templates that such settings are valid.

In this section:

§30.6.1 [Creating a template from a project configuration file](#) on page 862

§30.6.2 [Deciding what to include in a general configuration template](#) on page 862

§30.6.3 [Chaining configuration templates](#) on page 863

See also:

§4.1 [Working with Mif2Go configuration files](#) on page 91

§30.2 [Referencing configuration files and templates](#) on page 851

§33.1 [Using a different configuration for selected files](#) on page 919

30.6.1 Creating a template from a project configuration file

To create a general configuration template:

1. Copy one of your configuration files (one that has the most commonly used settings) to another directory, and give it a different name with extension `.ini`; for example `MyTemplate.ini`.
2. Delete from `MyTemplate.ini` any settings that apply *only* to the particular project from which you copied the configuration file. Also delete all macro definitions.
3. Delete from all your project configuration files any unused sections that have settings in `MyTemplate.ini`.
4. Delete from all your project configuration files any settings that occur in `MyTemplate.ini`, unless a setting has a different value. Settings in a project configuration file override those in a configuration template.
5. In your project configuration file, specify the following to reference the template:

```
[Templates]
; Configs = path to configuration template file
Configs=path\to\MyTemplate.ini
```

Because you originally copied `MyTemplate.ini` from your project configuration file, `MyTemplate.ini` still has a setting referencing the next configuration template in the chain supplied by **Mif2Go**; so the template chain remains unbroken.

The idea is to have as little as possible in individual project configuration files, and keep most common settings in the template. However, there are a few settings that can appear only in the project configuration file; see §30.6.2 [Deciding what to include in a general configuration template](#) on page 862.

30.6.2 Deciding what to include in a general configuration template

A configuration template should include settings and values that you normally use in most or all projects for a given type of output. The settings in the template file apply to any configuration file that references that template, reducing the need to add the same settings to every project configuration file.

Mif2Go supplies an extensive collection of templates already chained together. You can insert other templates in this chain, between your starting project configuration file and the first **Mif2Go**-supplied file in the chain. However, you might prefer to add settings to the appropriate editable configuration file supplied in the existing chain; see §30.5 [Deciding which configuration file to edit](#) on page 856.

Books that share a FrameMaker template probably can share the same **Mif2Go** configuration template. You might want different configuration templates for TOC, IX, and regular chapters. Configuration templates for different books might all reference a company-wide configuration template that specifies logos and other boilerplate items.

*Project overrides
template*

If a setting has a value in a template file that is different from its value in the project configuration file, the value in the project configuration file takes precedence, allowing you to override the template when necessary; see §33.1.2 [Understanding precedence of configuration settings](#) on page 919.

*Define macros
elsewhere*

Do not include macro definitions in a general configuration template; keep macro definitions in a separate library file; see §28.2.4 [Including macro definitions in your own macro library](#) on page 794 and §30.1.4 [Understanding how macro libraries are organized](#) on page 851.

<i>Specify run-time values elsewhere</i>	Do not include [UserVars] or [UserVarPrompts] in a configuration template; these two sections must be in your project configuration file. See §34.5 Supplying run-time values for user variables on page 941.
<i>Specify condition settings elsewhere</i>	Do not include [ConditionsShown] in a configuration template; this section must be in your project configuration file. See §5.4.1 Applying condition Show/Hide settings on page 123.
<i>Some settings are duplicated</i>	Although the settings in Table 30-8 can be included in a configuration template, some will end up in the project configuration file anyway; either because Mif2Go originates them, or because their values can be changed at run time via <i>Choose Project</i> dialog or <i>Export</i> dialog (or by Mif2Go). If you remove one of these settings from the project configuration file, Mif2Go will put it back in, at the end of the section where it belongs. If the section itself is missing, Mif2Go places the section and the setting near the end of the project configuration file.

Table 30-8 Configuration options determined at run time

Configuration section	Option	Reference
Automation	CompileHelp	35.10
	WrapAndShip	35.2
Setup	IDFileName	5.3.4.1
	PluginVersion	D.2.9
	PrjFileName	C.3
	UseExistingDCL	5.1.4
	UseExistingMIF	5.1.3
	WriteAllGraphics	5.7.2.1
	WriteEquations	5.7.2.1
Graphics	UseGraphicPreviews	5.7.2.2
	UseOriginalGraphicNames	5.7.2.3
HelpOptions	MakeCombinedCnt	8.2.8

30.6.3 Chaining configuration templates

A configuration template can include a setting for [Templates]Configs, specifying yet another template file. This allows you to create a chain of templates for **Mif2Go** to search for settings. The chain can be any length. All files in the chain must have distinct names; the search stops if **Mif2Go** finds a repeated template name. The settings in all templates in a chain are applied to your project configuration in a cascade, at run time.

<i>Precedence of templates</i>	In a chain of templates, if the same setting appears in more than one template file but has a different value in each file, the value for that setting in a template closer in the chain to the project configuration file overrides the value in any template farther away in the chain from the project configuration file; and a value for that same setting in the project configuration file overrides the closest template value. See §33.1.2 Understanding precedence of configuration settings on page 919.
--------------------------------	---

30.7 Applying FrameMaker conversion templates

When you convert documents from within FrameMaker, **Mif2Go** can use the FrameMaker **Import Formats** feature to temporarily apply a different template to your document for conversion purposes. Even if you have chapter files open in a FrameMaker book you are

converting, those files are not affected by **Mif2Go** importing formats for the conversion (but see §30.7.4 [Avoiding template-related disasters](#) on page 866).

When you start a project, you can use a **Mif2Go Set Up** dialog to specify which template to use, and which items to import. The *Set Up* dialogs present the same choices as the *FrameMaker Import Formats* dialog; see §3.4.1 [Importing formats from a FrameMaker template](#) on page 79. **Mif2Go** stores the results in the configuration file.

In this section:

- §30.7.1 [Specifying conversion-template settings](#) on page 864
- §30.7.2 [Applying alternate conversion templates](#) on page 865
- §30.7.3 [Changing template options](#) on page 866
- §30.7.4 [Avoiding template-related disasters](#) on page 866
- §30.7.5 [Troubleshooting template import problems](#) on page 866

See also:

- §2.4 [Importing formats from a conversion template](#) on page 67
- §3.4.1 [Importing formats from a FrameMaker template](#) on page 79
- §5.4 [Applying FrameMaker conditions and variables](#) on page 122
- §34.1.4 [Importing formats and conditional text settings](#) on page 936

30.7.1 Specifying conversion-template settings

To have **Mif2Go** apply a FrameMaker template to your document, your configuration file must include values for three template settings: `ApplyTemplateFile`, `TemplateFileName`, and `AppliedTemplateFlags`. For example:

```
[Templates]
; ApplyTemplateFile = No (default) or Yes (save and restore document)
ApplyTemplateFile=Yes
; TemplateFileName = filename.fm (FrameMaker template for export)
TemplateFileName="G:\Omnisys\UG\UGTplHlp.fm"
; AppliedTemplateFlags = 0 (default) or bitfield spec of properties
AppliedTemplateFlags=147
```

Unless the template to be applied is in the same directory as the document you are converting, specify a full absolute path (not a relative path) for `TemplateFileName`. Enclose the entire path in quotes if the path includes any spaces (and see §2.1 [Naming files, directories, and paths](#) on page 65).

FrameMaker uses the value of `AppliedTemplateFlags` to determine which items to import. This number is a representation of the choices you make via **Import Formats**: the decimal sum of the values (listed in [Table 30-9](#)) of all the options you check. For example, if you check paragraph formats, cross-reference settings, and conditional-text settings, the value of `AppliedTemplateFlags` would be $1+128+16=145$.

Note: A value of zero (`AppliedTemplateFlags=0`) means import *everything*. To import *nothing*, you must set `ApplyTemplateFile=No`.

Table 30-9 Template flag values for importing formats

Import Formats option	Import formats flag value	
	Decimal	Hexadecimal
Paragraph formats	1	0x0001
Character formats	2	0x0002
Page layouts	4	0x0004

Table 30-9 Template flag values for importing formats (continued)

Import Formats option	Import formats flag value	
	Decimal	Hexadecimal
Table formats	8	0x0008
Conditional text settings	16	0x0010
Reference pages	32	0x0020
Variable definitions (<i>both System and User</i>)	64	0x0040
Cross-reference settings	128	0x0080
Color definitions	256	0x0100
Math definitions	512	0x0200
Document properties (<i>at your own risk</i>)	1024	0x0400
Remove manual page breaks	16384	0x4000
Remove overrides	32768	0x8000

If absolutely necessary, you can also specify the following template option:

```
[Templates]
; ImportDocProps = No (default) or Yes (include doc props in import)
ImportDocProps=Yes
```

By default, **Mif2Go** instructs FrameMaker *not* to import document properties, which include the following:

- custom marker types
- footnote properties
- volume, chapter, page, paragraph, footnote, and table footnote number styles
- text options “allow line breaks” settings
- line layout “feather” settings

Note: *A defect in FrameMaker can cause numerous document settings to change unexpectedly when you import document properties, even from the same file.* Unless you have a very good reason to import document properties, use only the default setting: `ImportDocProps=No`.

30.7.2 Applying alternate conversion templates

If you are converting a FrameMaker book, and some files in the book use different templates, you must create:

- a separate configuration file for each of those FrameMaker files
- an (optional) alternate conversion template for each different FrameMaker template.

In each of the separate configuration files, specify the conversion template for **Mif2Go** to apply to the corresponding FrameMaker file.

For example, suppose you use a special template named `TitleTpl.fm`, located in the input directory, for the first file of your book, which is named `Titlepage.fm`. In the project directory you would create a configuration file named `Titlepage.ini` that contains these two lines:

```
[Templates]
TemplateFileName = TitleTpl.fm
```

See §33.1 [Using a different configuration for selected files](#) on page 919 for information about creating separate chapter-specific configuration files.

30.7.3 Changing template options

If you are converting to HTML or XML, you can use the following roundabout method to change the template choices in an existing project configuration file:

1. Rename the current project configuration file, or move it to another directory.
2. Choose **Set Up Mif2Go Export...** again from the FrameMaker **File** menu, and make the template settings you want this time. When you click **OK**, a new project configuration file pops up in Notepad, with an `AppliedTemplateFlags` value that corresponds to your template settings.
3. Highlight the line that starts with `AppliedTemplateFlags=` and copy it.
4. Open your original (renamed or moved) configuration file in the same Notepad, and paste the new `AppliedTemplateFlags` line over the existing line.
5. **Save As** `_m2html.ini` (or whatever name the original project configuration file had), replacing the new copy.

Note: If you did not specify importing formats when you set up the project, you must also supply values for `ApplyTemplateFile` and `TemplateFileName`; see §30.7.1 [Specifying conversion-template settings](#) on page 864.

30.7.4 Avoiding template-related disasters

Mif2Go automatically saves your document before importing a template. If **Mif2Go** encounters a problem during template import, and you choose to stop the conversion and investigate, do not assume that the import did not happen; some part of the import always has already taken place. Therefore, *do not* fix up and save the problem file; instead, just look at the file to determine the cause of the problem, then close the file *without saving it*. After that you can open the original file, fix the problem, and rerun **Mif2Go**.

When you specify `[Setup]GenerateBook=Yes`, the same applies to *all files in the book that precede the problem file*. It is not safe to just rerun **Mif2Go**, because all those modified files would be re-saved over your originals! Close them all, *without saving*.

If you are importing a template into a book, it is a good idea to exit FrameMaker, without saving files, immediately after **Mif2Go** finishes. That way your standard procedure can be to close *all* files without saving, regardless of whether **Mif2Go** completes the conversion successfully or stops in the middle upon encountering a problem.

30.7.5 Troubleshooting template import problems

If it looks as though your FrameMaker template is not being applied, the problem could be one of the following:

- [Wrong path to the template file](#)
- [Missing or incorrect option](#)
- [Incorrect flag number](#)
- [Template file problem.](#)

*Wrong path to the
template file*

The path to the template file might be wrong, resulting in an incorrect value for `[Setup]TemplateFileName`; or the value might specify a relative path. Unless the template file is in the same directory as the document you are converting, you must specify an absolute path.

*Missing or
incorrect option*

Your configuration file might be missing a setting for `[Setup]ApplyTemplateFile`, or this option might be set to `No` instead of `Yes`.

- Incorrect flag number* The value of [Setup]AppliedTemplateFlags might be incorrect, resulting in **Mif2Go** importing the wrong set of formats from the template file.
- Template file problem* The template file itself might contain errors that prevent **Mif2Go** from opening it, such as a missing-font error.
- See §30.7.1 [Specifying conversion-template settings](#) on page 864.

31 Working with graphics

This section tells how to export and convert the graphics in your FrameMaker document, and control their appearance in the output produced by **Mif2Go**. Topics include:

- §31.1 [Choosing an appropriate graphics format](#) on page 869
- §31.2 [Converting and exporting graphics](#) on page 871
- §31.3 [Replacing and relocating graphics files](#) on page 887
- §31.4 [Specifying custom settings for individual graphics](#) on page 895
- §31.5 [Controlling image appearance in RTF output](#) on page 898
- §31.6 [Converting graphics with Microsoft Word filters](#) on page 904

See also:

- §5.7 [Processing graphics](#) on page 126
- §6.14 [Managing graphics for print RTF](#) on page 186
- §8.6 [Managing graphics for WinHelp](#) on page 263
- §23 [Including graphics in HTML](#) on page 703
- §35.7 [Placing graphics files for distribution](#) on page 965

31.1 Choosing an appropriate graphics format

Some graphics formats work better than others in each kind of output. For best appearance, you might want to convert graphics in your FrameMaker document to a more appropriate format. In some cases there is no choice: the graphics have to be converted. You do not have to alter graphics in the FrameMaker file itself; however, you might have to prepare an alternate set of graphics files, and set some configuration-file options.

In this section:

- §31.1.1 [Graphics formats for Word documents](#) on page 869
- §31.1.2 [Graphics formats for WinHelp](#) on page 869
- §31.1.3 [WMF format limitations](#) on page 870
- §31.1.4 [Graphics formats for HTML](#) on page 871

31.1.1 Graphics formats for Word documents

- WMF or BMP* Use WMF or BMP for bitmap graphics for Word. Although graphics destined for print look best at a resolution of at least 300 DPI (for low-end laser printers) and up to 1,200 DPI, you gain nothing by trying to increase the DPI of an existing bitmap graphic in FrameMaker; the upper limit of resolution is the DPI of the original graphic. If you choose WMF, see §31.1.3 [WMF format limitations](#) on page 870.
- 256 colors* Word graphics typically use 256 colors, sometimes more. It is best to stick with 256 colors, because the size increase for 24-bit color (“true color”, the only other real option) is more than 10 times, which can make files too big for Word to load.

31.1.2 Graphics formats for WinHelp

- WMF or BMP* WinHelp graphics must be in WMF or BMP format. Graphics are viewed at screen resolution, typically 96 DPI. Normally you want to use WMF, because WMF graphics have a much sharper image than BMP graphics. However, the WMF format has some drawbacks; see §31.1.3 [WMF format limitations](#) on page 870.

For very large graphics, BMP can be a better choice; see §31.2.1.1.3 [Specifying BMP instead of WMF graphics](#) on page 872.

*256 colors for
WinHelp 4*

WinHelp 4 allows 256-color bitmaps; the added space for 256 colors is relatively small, and the graphics usually look better than with 16 colors. WinHelp 3 allows only 16-color bitmaps (unless you use add-on DLLs). For any WinHelp use, 24-bit color (“true color”) is a very bad idea; it often crashes the Help Compiler during processing.

31.1.3 WMF format limitations

WMF graphics are like scripts for the Windows GDI. A WMF graphic can include vector elements (as in FrameMaker native graphics), text, and bitmaps (with up to 24-bit resolution). You can import WMF graphics into FrameMaker, where they are seen as Frame Vector facets. However, FrameMaker special-ungroup command `Esc g U` does not make it possible to edit a WMF graphic.

In WinHelp, WMF graphics can cause system crashes on Windows 9x or Windows ME; see §8.6.2 [Avoiding the GDI resource leak](#) on page 264 for more information.

The WMF graphic format has limitations:

- [Bezier curves become polylines](#)
- [Dashed line width is ignored](#)
- [Cropped images show all in WinHelp](#)
- [Fonts are not embedded](#)

*Bezier curves
become polylines*

WMF does not support Bezier curves (smoothed polylines in FrameMaker). WMF does support ellipses and elliptic arcs, with radii or chords available for the arcs. Bezier curves are represented by polyline segments. When **Mif2Go** converts FrameMaker vector graphics to WMF, **Mif2Go** generates the polyline segments; but some segments might be only one pixel long, if that is what it takes to emulate a Bezier display. The segmented polylines are indistinguishable from the original Bezier curves when viewed on screen. There is a difference in print, but you would have to use a magnifier to see it.

*Dashed line width
is ignored*

For dashed lines and other non-solid lines, MicroSoft code for WMF images sets line thickness to 1 (one), which is nominally 1.0 twip (1/20 pt) or 0.01 mm. This value is affected by scaling, so you always get the thinnest line drawable. If you try to use a different thickness for a non-solid line, you get the thickness you specified, but the line becomes solid. However, when you specify **Mif2Go** native graphics export to WMF, **Mif2Go** preserves the line style rather than the line width. To preserve both line style and line width, you would have to specify FrameMaker graphics export instead, and convert the image to a BMP instead of to a WMF. See §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883.

*Cropped images
show all in
WinHelp*

WMF does not support cropping. In FrameMaker, you can use the anchored frame to crop a large graphic to the area of interest. This does not work at all in WinHelp. Although you see the part you want in WinHelp, the rest of the image is also visible, running right over any nearby text or other graphics. This is true of both bitmaps and vectors. We advise cropping bitmaps to the actual displayed area before importing them into FrameMaker, even if you are importing by reference.

*Fonts are not
embedded*

WMF cannot embed fonts, and so relies on the viewing application to have the fonts available for rendering. If Windows substitutes another font with different metrics, the result can be quite ugly. WMF has no concept of text frames with flowing content. Instead, each line is an independent text line that cannot include any font-property changes, such as bold or subscript. Such changes require starting a new text line; however, you do not know where the last segment ended (one of the key differences between WMF and direct

Windows GDI calls), so you must compute the position based on your own estimate of the font metrics. A miscalculation (or a different font) can result in big gaps, or in overlapping text.

31.1.4 Graphics formats for HTML

JPEG, GIF, PNG Graphics formats that work well on the Web are JPEG, PNG, and GIF. We suggest JPEG for Web use. JPEG is universally supported by browsers, and we have yet to see an instance of a graphic where it behaved badly compared to other formats. Other formats might be useful in particular situations, but they are not universally supported by Web browsers.

See also:

§23 [Including graphics in HTML](#) on page 703

31.2 Converting and exporting graphics

A graphic image in your FrameMaker document might or might not be easy for **Mif2Go** to include in the output with reasonable quality, depending on a number of factors, including whether the image is:

- a bitmap graphic or a vector graphic
- embedded in your FrameMaker document or imported by reference
- in a format **Mif2Go** can handle well.

In this section:

§31.2.1 [Converting bitmap graphics](#) on page 871

§31.2.2 [Converting vector graphics](#) on page 874

§31.2.3 [Exporting and converting embedded graphics](#) on page 877

§31.2.4 [Exporting images and creating files from OLE objects](#) on page 881

§31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883

§31.2.6 [Embedding bitmap graphics in WMF for WinHelp](#) on page 886

§31.2.7 [Exporting embedded graphics imported from Word](#) on page 886

31.2.1 Converting bitmap graphics

The biggest issue for converting bitmap graphics has to do with resolution; this affects how you scale graphics, and how you handle fine detail in them:

- If you are converting to WinHelp or HTML, your graphics will be viewed at screen resolution, typically 96 DPI.
- If you are converting to print RTF, the graphics might be printed out at a much higher resolution.

In this section:

§31.2.1.1 [Converting bitmap graphics for WinHelp](#) on page 871

§31.2.1.2 [Converting bitmap graphics for print RTF](#) on page 873

§31.2.1.3 [Specifying external vs. internal metafiles for RTF output](#) on page 873

§31.2.1.4 [Converting bitmap graphics for Web use \(HTML\)](#) on page 874

31.2.1.1 Converting bitmap graphics for WinHelp

In this section:

§31.2.1.1.1 [Scaling \(or avoiding scaling\) screenshots](#) on page 872

[§31.2.1.1.2 Specifying WMF graphics as replacements](#) on page 872

[§31.2.1.1.3 Specifying BMP instead of WMF graphics](#) on page 872

31.2.1.1.1 Scaling (or avoiding scaling) screenshots

Screenshots do not scale well at all, not even a little. The text is messed up by even the slightest rescale.

If you use a screenshot graphic at its original size, unless it is only a button or a small dialog the graphic tends to overwhelm the accompanying text. If the graphic is a full-panel screenshot, it looks huge. And if you scale it at all in FrameMaker, any screen-capture text becomes illegible. You cannot reduce graphic size even by 5% and retain legibility.

These are your choices:

- Crop big images to show just the part you need, in a graphics editor (such as Paint Shop Pro, Photoshop, or Graphic Workshop; see [§5.7.2.3 Using third-party graphics converters](#) on page 130).
- Eliminate the screenshots entirely; if users are looking at Help, they also have the real application right there; you can tell them how to get to the screen you are discussing.
- Use thumbnails: little images that each link to a bigger version that is typically displayed as a pop-up in its own window.
- If you are preparing bitmaps for WinHelp use, resample them so that they are at screen resolution, typically 96 DPI, at the size at which you wish to display them.

See also [§31.5.1 Rescaling bitmap graphics](#) on page 898.

31.2.1.1.2 Specifying WMF graphics as replacements

If you convert graphics outside of **Mif2Go**, you can direct **Mif2Go** to use the replacement graphics files.

Suppose, for example, you use a third-party graphics tool to convert imported-by-reference TIFF images in your document to WMF, and place the WMF files in the same directory as the original TIFF files. You would specify the following options in your project configuration file:

```
[Graphics]
FileNames=Map

[GraphFiles]
tif=wmf
```

When you choose **File > Save Using Mif2Go...** in FrameMaker, **Mif2Go** reads in the WMF graphics, adds any FrameMaker elements (such as callouts) that are in the illustrations, and rewrites the graphics as scaled .wmf files, which are referenced in the .rtf files **Mif2Go** produces for WinHelp.

For more information about generating WMF graphics for WinHelp, see:

[§8.6.2 Avoiding the GDI resource leak](#) on page 264.

[§31.2.6 Embedding bitmap graphics in WMF for WinHelp](#) on page 886.

[§31.3.2 Changing graphics files for RTF output](#) on page 890.

[§31.3.2.3 Using different bitmaps for print RTF and for WinHelp](#) on page 894.

31.2.1.1.3 Specifying BMP instead of WMF graphics

Sometimes when you display and scroll a large bitmap graphic (around 300 KB) in a Help file, Windows 9x resources drop to a point where processing halts. This is a Windows GDI defect acknowledged by Microsoft. It happens with WMF graphics only, not with BMP graphics; and only on Windows 9x systems, not on Windows NT or Windows 2000.

The solution is to use BMP graphics instead of WMF graphics in your Help file:

- If you are using the FrameMaker export filters, specify BMP graphics instead of the default WMF:

```
[Setup]
GraphicExportFormat=BMP
```

- If you are using [Graphics]FileNames=Map, change the setting in [GraphFiles] to match:

```
[GraphFiles]
wmf=bmp
```

- If you are producing graphics some other way, make sure you are creating .bmp files.

To prevent **Mif2Go** from making your graphics into .wmf files anyway, also set:

```
[Graphics]
EmbedBMPsInWMFs=No
```

31.2.1.2 Converting bitmap graphics for print RTF

If your FrameMaker document includes imported-by-reference bitmap graphics destined for print documents, in formats such as TIFF, **Mif2Go** can use the FrameMaker graphics export filters (see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883) to create WMF graphics, then integrate the graphics into RTF files. The disadvantage is that you get only screen resolution, which is not the best for print quality. You can increase the DPI, but all that buys you is a larger image, not improved resolution. This is true even if the original bitmap is at a higher resolution. See §6.14.2 [Converting referenced graphics](#) on page 187.

*Better print quality
via third-party
converters*

For improved print quality, use a third-party tool such as Graphic Workshop to convert bitmap graphics from other formats to WMF or BMP. You get significantly better image quality with a pixel-to-pixel conversion via a third-party converter than you would with the re-rendering process used by the FrameMaker export filter. See §5.7.2.3 [Using third-party graphics converters](#) on page 130.

*Callouts and
compound
graphics*

Some of your graphics in formats other than WMF or BMP might include callouts and other bits and pieces created with FrameMaker drawing tools. For these, your only choice *at conversion time* is to use the FrameMaker export filters to make WMFs. **Mif2Go** integrates the resulting WMF graphics into the Word RTF files.

*If you do not
convert graphics*

Only WMF and BMP graphics can be embedded in Word. If you provide only a GIF image (for example), **Mif2Go** embeds an INCLUDEPICTURE field (Word 8) or IMPORT field (Word 7) instead, with the name of the GIF file. Then, when you load the RTF file in Word, Word reads the GIF file and uses its own filters to convert the image to BMP internally.

See §6.14 [Managing graphics for print RTF](#) on page 186.

31.2.1.3 Specifying external vs. internal metafiles for RTF output

If you are converting to RTF, there are two ways to include a WMF file:

- directly in RTF: an “internal” metafile
- as a separate file referenced from RTF: an “external” metafile.

For WinHelp, metafiles must be external.

For Word, metafiles should be internal. Word loses track of external graphics that are referenced using relative paths. These are the default metafile types:

<u>Output type</u>	<u>Output option</u>	<u>Metafile type</u>
Print RTF	Standard	Internal
WinHelp	Help	External

You can override the default by specifying the metafile type yourself:

```
[Graphics]
; Metafiles = Internal (in place) or External (in AAAAAAnnn.WMF)
; always use External for winhelp; Internal is better for Word
Metafiles=Internal
```

External metafiles are named using the first five characters of the RTF file name, then a three-digit number, then the .wmf extension. See §5.7.4.3 [Naming external graphic metafiles](#) on page 134 for more information.

31.2.1.4 Converting bitmap graphics for Web use (HTML)

Graphics formats that work best in FrameMaker for printed documents generally are not those that work well on the Web. If your graphics are not already in a Web-ready format, you have two basic choices:

- Have **Mif2Go** use FrameMaker export filters to produce JPEG, PNG, or other Web-usable formats (see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883). This is the easiest way, and the default way for HTML.
- Handle conversion outside of **Mif2Go** with a third-party tool such as Graphic Workshop (see §5.7.2.3 [Using third-party graphics converters](#) on page 130).

If the original graphics in your FrameMaker document are in a Web-ready format, and each is alone in its frame, you can use them *as is*; see §31.3.1.5 [Including referenced graphics without converting](#) on page 889.

See also:

§23 [Including graphics in HTML](#) on page 703

31.2.2 Converting vector graphics

Vector graphics can be rescaled without losing image quality; resolution is not a concern. The illustrations you create using FrameMaker graphics tools are vector graphics.

In this section:

§31.2.2.1 [Converting FrameMaker vector graphics to RTF](#) on page 874

§31.2.2.2 [Converting FrameMaker vector graphics to HTML](#) on page 875

§31.2.2.3 [Converting EPS graphics](#) on page 875

31.2.2.1 Converting FrameMaker vector graphics to RTF

For RTF output, **Mif2Go** converts native FrameMaker vector graphics into Windows Metafile (WMF) format, to produce metafiles. The metafiles are acceptable to MS Word, to the WinHelp compiler, and to numerous other applications. These particular WMFs are not bitmaps; they are scalable and editable vector graphics.

If the FrameMaker vector graphics in your document include imported bitmaps, those bitmap files must be WMF or BMP. Otherwise you must create or convert them to WMF or BMP equivalents, and map them using settings in configuration section [GraphFiles]; see §31.3.2 [Changing graphics files for RTF output](#) on page 890.

31.2.2.2 Converting FrameMaker vector graphics to HTML

Although you can use these WMFs for HTML, they will be viewable only with Internet Explorer, and not with other browsers. To create the WMFs, you must use **Mif2Go** in a preliminary graphics exporting step ostensibly directed to WinHelp RTF, then start over and target HTML.

For HTML output viewable with browsers other than Internet Explorer, you must direct **Mif2Go** to use the FrameMaker export filters to convert FrameMaker vector graphics, and specify the output format you want; see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883 for the available options.

See also:

§23 [Including graphics in HTML](#) on page 703

31.2.2.3 Converting EPS graphics

EPS can be a difficult format to work with. An EPS graphic has two parts: a PostScript image used only for printing, and a preview image (deliberately low resolution) intended only for viewing on screen:

- PostScript:* This is what you print from FrameMaker, at least when you print to a PostScript printer.
- Preview:* Usually TIFF, but could be in other formats such as PICT on the Mac, or WMF in Windows. Sometimes there is a FrameImage facet also.

EPS graphics present problems for some programs; a lot depends on the format of the preview image.

In this section:

- §31.2.2.3.1 [Deciding how to treat EPS graphics](#) on page 875
- §31.2.2.3.2 [Including only the preview image](#) on page 876
- §31.2.2.3.3 [Including both preview and EPS images](#) on page 876
- §31.2.2.3.4 [Replacing EPS graphics](#) on page 876
- §31.2.2.3.5 [Referencing EPS graphics in Word](#) on page 877

31.2.2.3.1 Deciding how to treat EPS graphics

- | | |
|---------------------------------------|--|
| <i>Export embedded EPS</i> | If EPS graphics are embedded in your FrameMaker document, by default Mif2Go exports the graphics to create external .eps files. You can run the conversion once to export the EPS graphics, convert the graphics to another format outside of Mif2Go , then run the conversion again, this time directing Mif2Go to use the already converted external files. In the final conversion output, Mif2Go can replace references to the EPS graphics with references to the matching files; see §31.2.2.3.4 Replacing EPS graphics on page 876. |
| <i>Export EPS preview</i> | Another alternative is to use the FrameMaker graphic export filters (see §31.2.5 Converting graphics with FrameMaker export filters on page 883). These filters do a terrible job because they start off with the low-quality preview and go downhill from there; the EPS preview is meant only for identification of the graphic, not for actual use. |
| <i>Convert using third-party tool</i> | For higher quality, you can use a third-party graphics tool (see §5.7.2.3 Using third-party graphics converters on page 130), and convert external EPS files (either referenced graphics or embedded graphics exported by Mif2Go from your document) to matching RTF- or HTML-compatible graphics: <ul style="list-style-type: none"> • BMP or WMF for RTF • GIF, JPEG, or PNG for HTML. |

However, most graphics tools convert only the preview image. To make a better rendering from the PostScript part, you need a converter that can interpret PostScript. You could use GhostScript, which is a free PostScript interpreter:

<http://www.cs.wisc.edu/~ghost/>

with the free converter ImageMagick:

<http://www.imagemagick.org/>

*Use EPS files as
is for Word*

For print RTF, as an alternative you could set [Graphics]EpsiUsage=Retain (see §31.2.2.3.5 [Referencing EPS graphics in Word](#) on page 877), and keep the EPS files with the RTF output until Word loads the RTF file; then Word will import the EPS image itself. Unfortunately, this method does not preserve the FrameMaker scaling. Unless the original EPS file was imported into Frame Maker at 100% scale, you will have to change the size in Word after loading the RTF produced by **Mif2Go**. You will still see only the ugly preview on screen, but the graphic will print nicely—at least on a PostScript printer. For more information, see §31.6 [Converting graphics with Microsoft Word filters](#) on page 904.

31.2.2.3.2 Including only the preview image

If the graphic includes a preview facet **Mif2Go** understands, such as FrameImage or WMF, by default **Mif2Go** places that image in the converted file and discards the PostScript. If you are creating WinHelp or HTML, that can be adequate.

If you are converting to RTF you can direct **Mif2Go** to use the FrameImage, if one is present:

```
[Graphics]
; UseFrameImage = No (default)
; or Yes (in preference to other formats)
UseFrameImage=Yes
```

31.2.2.3.3 Including both preview and EPS images

If you want a better rendition than the screen-resolution preview image, set the following option:

```
[Graphics]
; EpsiUsage =
; Preview (only, default),
; EPS (no preview), or
; Retain (both)
EpsiUsage=Retain
```

Options for EpsiUsage have the following effects on EPS graphics:

Retain	The preview image is converted along with the PostScript image.
EPS	Only the PostScript is converted; if you are converting to Word RTF, Word displays only a gray box, although the image will print correctly on a PostScript printer.
Preview	Only the preview image is converted; you lose the PostScript version.

31.2.2.3.4 Replacing EPS graphics

For many kinds of output you will want to convert EPS graphics to another format. For example, for WinHelp the graphics must be in WMF or BMP format; for HTML, graphics should be JPEG, GIF, or PNG. For Word, you will have to convert EPS graphics if you want to be able to scale the images.

If the EPS graphics are embedded in your FrameMaker document, the default settings in section [GraphExport] make external .eps files from them; see §31.2.3 [Exporting and converting embedded graphics](#) on page 877.

You must do the following:

1. Convert the .eps files to an appropriate format, using a third-party tool; see §31.2.2.3.1 [Deciding how to treat EPS graphics](#) on page 875.
2. Map the graphics to the new format.

- For Word or WinHelp:

```
[Graphics]
FileNames=Map
```

```
[GraphFiles]
eps=wmf
```

- For HTML:

```
[Graphics]
GraphSuffix = jpg
```

3. Specify the location of the replacement files.

- For Word or WinHelp, set FilePaths=Retain if the new graphics are in the same directory as the .eps files, or FilePaths=None if you put them in the project directory:

```
[Graphics]
FilePaths = Retain
```

See §31.3.2 [Changing graphics files for RTF output](#) on page 890.

- For HTML, if the new graphics are in the project directory:

```
[Graphics]
StripGraphPath = Yes
```

Or, you can tell **Mif2Go** exactly where you put the new graphics:

```
[Graphics]
GraphPath = relative/path/to/graphics/files
```

The path you specify for GraphPath should be relative to the wrap directory (see §35.3 [Understanding path values for deliverables](#) on page 957). This path will be used in HTML output, as the relative path from the HTML files to their referenced graphics. This option sets the src attribute of the tags; it does not change the location of the graphics files themselves. See §23.3 [Locating graphics files for HTML](#) on page 704. To copy files to another location, see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.

31.2.2.3.5 Referencing EPS graphics in Word

If the lack of scaling is not an issue, you can let Word import an EPS graphic. The default [GraphExport] settings make the embedded file an external .eps, which is named in a Word INCLUDEPICTURE field (Word 8) or IMPORT field (Word 7) when you run **Mif2Go**. The result is that you see the TIFF preview in Word. If you print to a PostScript printer you see the real EPS; however, if you print to any other printer, you see only the TIFF preview.

31.2.3 Exporting and converting embedded graphics

When you import images into FrameMaker by *copying* (instead of by *reference*), the images are embedded in your FrameMaker file. By default, **Mif2Go** exports most types of embedded images to create external graphics files.

In this section:

- §31.2.3.1 [Understanding how embedded graphics are exported](#) on page 878
- §31.2.3.2 [Replacing embedded graphics](#) on page 878
- §31.2.3.3 [Converting embedded graphics for Word without exporting](#) on page 879
- §31.2.3.4 [Exporting embedded graphics before converting](#) on page 879
- §31.2.3.5 [Setting export options for all embedded graphics](#) on page 880
- §31.2.3.6 [Setting export options for each embedded graphic type](#) on page 880

See also:

- §5.7.3.2 [Processing embedded graphics separately](#) on page 132
- §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134
- §31.2.7 [Exporting embedded graphics imported from Word](#) on page 886

31.2.3.1 Understanding how embedded graphics are exported

Mif2Go uses an internal graphic export process to create external files from graphics embedded in your FrameMaker document, provided those graphics do not include, in the same anchored frame, other elements such as callouts, titles, or additional images. You get the original format at its full original resolution. You do *not* get the original name, because FrameMaker discards the original name when the image is embedded; there is no way to recover that name. Therefore, **Mif2Go** assigns a computed name to each exported file; see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134.

If an image to be exported is *not* alone in its anchored frame, **Mif2Go** must use FrameMaker export filters instead; see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883. In that case, callouts, montages, and so forth, are retained, but the resolution is generally much worse than the resolution of the original image.

31.2.3.2 Replacing embedded graphics

If you have access to the original image files, the best option is simply to replace the images in FrameMaker. This process can be labor intensive, but you have to do it only once. With each embedded graphic selected in FrameMaker, re-import the original graphic file by reference; the referenced graphic replaces the embedded graphic. Make sure you select the embedded image itself (not the anchored frame) before you import the replacement. See §2.5.2 [Planning for graphics processing](#) on page 69.

If you do not have access to the original image files, you can still export embedded graphics and bring them back in as referenced graphics, either with the file names assigned by **Mif2Go** (see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134), or after changing the assigned names to more suitable names. See §5.7.3.2 [Processing embedded graphics separately](#) on page 132.

If your document includes only a few graphics, you can match them visually with their embedded counterparts. If your document includes a great many graphics, the challenge will be to get them back into the correct anchored frames without scrutinizing each image individually.

The ASCII DCL produced when you export the embedded graphics includes the UniqueIDs of both the anchored frame and the image in each case, along with the assigned file name. Also included are the rotation and DPI of each image.

For example, exporting a single embedded graphic from FrameMaker file `FigsTbls.fm` produces a file named `Figs001.gif`, and this description in the ASCII DCL output:

```

(layout fr_def 1)
(text hyper_tok loc = 999590)
(layout anchored 20 15)
(layout frame pos inline)
(layout frame within col_hor)
(layout fr_size page + 226233 257103 407900 270400)
(layout fr_line pattern solid)
(layout fr_line color black)
(layout fr_line thick 100)
(layout fr_pen pattern invisible)
(layout fr_fill pattern invisible)
(layout fr_fill color black)
((graph obj external)
(text hyper_tok loc = 1042546)
(graph line pattern solid)
(graph line color black)
(graph line thick 100)
(graph pen pattern invisible)
(graph fill pattern back)
(graph fill color black)
(layout fr_size frame + 28000 5000 347916 202083)
(layout frame rotation = 0)
(graph ras_dpi in = 96)
(dcl include graphic 'Figs0001.gif')
)
)

```

You can identify the location in the FrameMaker file using the (text hyper_tok loc = . . .) controls. You see the UniqueIDs of both the anchored frame (999590) and the image itself (1042546).

Unless the embedded graphics include callouts or other added elements, you have all the information you need to re-import by reference without loss, using a third-party scripting tool.

31.2.3.3 Converting embedded graphics for Word without exporting

You can get embedded graphics out of FrameMaker and into the output *without* exporting them as separate file only if both of the following are true:

- The document output type is Word RTF.
- The embedded graphics are in WMF, BMP, or FrameImage format.

Mif2Go converts these embedded graphics and includes them directly in the Word output.

31.2.3.4 Exporting embedded graphics before converting

By default, **Mif2Go** exports from your FrameMaker document all embedded graphics (except WMF, BMP, or FrameImage), and saves them as separate external graphics files. **Mif2Go** exports the graphics in their original format, whatever that was; there is no setting to change the format.

If the graphics are not already in an appropriate format, you can inspect the exported files, if necessary alter or replace them, and use configuration-file settings to map the old embedded graphics to the newly exported graphics files. This can make your conversion project a three-step process:

1. Run **Mif2Go** to export embedded graphics as separate files; see §31.2.3.5 [Setting export options for all embedded graphics](#) on page 880.
2. Examine the exported files. If the graphics are not in an appropriate format, or you want to rescale them, use a third-party graphics program to alter or replace the

graphics; see §5.7.2.3 [Using third-party graphics converters](#) on page 130. Or, you can try having **Mif2Go** use FrameMaker export filters to convert them; see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883.

3. Run **Mif2Go** again, specifying a mapping from the embedded graphics to the alternate graphics; see §31.3 [Replacing and relocating graphics files](#) on page 887.

Even if you are not going to use the graphics, exporting them first speeds up processing of the rest of your document.

If you are going to use the graphics, name clashes are likely; see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134. It is better to take the newly exported graphics files, give them proper names, and go back into FrameMaker and import them *by reference* in place of the original embedded images; then run the conversion again. Think of it as a way to atone for past sins of importing by copying.

31.2.3.5 Setting export options for all embedded graphics

To export embedded graphics from your document:

```
[GraphExport]
; make external files when they need to be converted or changed
; normally wmf, bmp, and rf files do *not* need to be changed
; the first sets the default for the rest
; ImportGraphics =
;   Normal (default),
;   Retain (in file), or
;   Export (external files)
ImportGraphics = Export
```

Settings for ImportGraphics have the following effects on your document when you run **Mif2Go**:

Normal	Retains embedded graphics of some types without exporting them, as indicated by the default listed for each type in §31.2.3.6 Setting export options for each embedded graphic type on page 880.
Retain	Does not export any embedded graphics.
Export	Exports all embedded graphics.

Normally, **Mif2Go** does not export BMP, WMF, or FrameImage (RF) graphics, because these types can be converted successfully for RTF output without creating external files. **Mif2Go** exports all other types by default, so you can use a third-party graphics conversion tool to convert the images to a usable format. See §5.7.2.3 [Using third-party graphics converters](#) on page 130.

To produce the WMF images required for RTF output (Word or WinHelp), **Mif2Go** can use *only* BMP, WMF, and RF embedded graphics; if your document contains embedded graphics in other formats, you must provide **Mif2Go** with replacement images.

31.2.3.6 Setting export options for each embedded graphic type

To replace a graphic of a type that is *not* exported by default, you must specify that its type should be exported. For example, if you want to export BMP graphics, specify these settings:

```
[GraphExport]
ImportGraphics=Normal
ExportBmpFiles=Yes
```

These are the graphic types for which you can set an individual export option, and the default value of the option for each type:

```
[GraphExport]
; ExportWmfFiles = Yes (makes external .wmf files) or No (default)
ExportWmfFiles=No
; ExportBmpFiles = Yes (makes external .bmp files) or No (default)
ExportBmpFiles=No
; ExportRfFiles = Yes (makes external .rf files) or No (default)
ExportRfFiles=No
; ExportPctFiles = Yes (makes external .pct files, default) or No
ExportPctFiles=Yes
; ExportTifFiles = Yes (makes external .tif files, default) or No
ExportTifFiles=Yes
; ExportGifFiles = Yes (makes external .gif files, default) or No
ExportGifFiles=Yes
; ExportJpgFiles = Yes (makes external .jpg files, default) or No
ExportJpgFiles=Yes
; ExportPngFiles = Yes (makes external .png files, default) or No
ExportPngFiles=Yes
; ExportPcxFiles = Yes (makes external .pcx files, default) or No
ExportPcxFiles=Yes
; ExportWpgFiles = Yes (makes external .wpg files, default) or No
ExportWpgFiles=Yes
; ExportCdrFiles = Yes (makes external .cdr files, default) or No
ExportCdrFiles=Yes
; ExportEpsFiles = Yes (makes external .eps files, default) or No
ExportEpsFiles=Yes
```

31.2.4 Exporting images and creating files from OLE objects

Mif2Go does not support OLE or OLE2; these are proprietary, undocumented Microsoft Structured Storage formats. An OLE object is like a binary mini-file system. The “files” in an OLE object contain information needed for editing by the application that originally created the object. The application is not required for viewing in FrameMaker or printing, because at least one of the “files” is a WMF image that can be used for display (but only on a Windows system).

In this section:

§31.2.4.1 [Extracting the WMF preview image from an OLE object](#) on page 881

§31.2.4.2 [Extracting and exporting all WMF images from an OLE object](#) on page 882

§31.2.4.3 [Extracting OLE images with FrameMaker export filters](#) on page 882

§31.2.4.4 [Retrieving OLE objects for use in the original application](#) on page 882

31.2.4.1 Extracting the WMF preview image from an OLE object

Each OLE object contains one or more WMF images. One of these is the preview image you see in FrameMaker; we guess it is the last WMF in the object, and that is what **Mif2Go** extracts, by default:

```
[GraphExport]
; ImportGraphics = Normal (default), Retain (in file)
; or Export (ext files)
ImportGraphics=Normal
; ExportWMFFiles=Yes (makes external .wmf files) or No (default)
ExportWMFFiles=Yes
```

When `ImportGraphics=Normal`, **Mif2Go** exports the extracted WMF preview image as a .wmf file; otherwise the image is not exported.

When `ImportGraphics=Export`, **Mif2Go** does *not* hunt for a preview WMF image in an OLE objects and merely dumps out the object. Our experience has been that such objects cannot be converted to anything useful. However, you might find that by changing the `.ole` extension to the extension required by the application that produced the object, the file can be opened and edited in the original application; see §31.2.4.4 [Retrieving OLE objects for use in the original application](#) on page 882.

31.2.4.2 Extracting and exporting all WMF images from an OLE object

With the following settings, **Mif2Go** extracts *all* WMFs from an OLE object, and exports each to a separate `.wmf` file:

```
[GraphExport]
ImportGraphics=Normal
ExportWMFFiles=Yes
; MultipleOLE = No (export only the last WMF image in OLE object), or
; Yes (export all WMFs, with "Xn" suffixes for all but the last)
MultipleOLE=Yes
```

Set `MultipleOLE=Yes` only if the single WMF that **Mif2Go** extracts by default turns out to be the wrong image.

*How WMFs from
OLE objects are
named*

When `ExportWMFFiles=Yes` and `MultipleOLE=Yes`, each WMF image extracted from an OLE object becomes a `.wmf` file. Mostly you will get only two or three WMFs. The last WMF extracted is the default file, and keeps the original graphic file name; for example, `aa123456.wmf` (see §5.3 [Identifying files and objects](#) on page 117). All other files extracted from the object get the same name with the addition of a base-name suffix `Xn`, where `n` is 1, 2, ..., 10, ... 100, and so forth; for example, `aa123456X2.wmf`. If one of these other files turns out to be the correct image, you must delete the incorrect default `.wmf` file, and rename the correct file by removing the `Xn` suffix.

31.2.4.3 Extracting OLE images with FrameMaker export filters

Having **Mif2Go** extract WMFs works for the majority of OLE objects, but not for all of them. If you still do not get a satisfactory image, you might have to resort to FrameMaker export filters, which use the displayed image to create a graphic **Mif2Go** can use; see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883.

The resulting file will be large. If you are converting to Word, the embedded graphics in RTF files are usually not compressed, and even those that can be compressed (256-color or less) do not compress well. If Word can load the RTF at all, you might be able to save in Word `.doc` format to get a more manageable size; this format has better compression. See §6.14.11 [Embedding graphics in converted RTF files](#) on page 193.

31.2.4.4 Retrieving OLE objects for use in the original application

If you use **Mif2Go** to export from FrameMaker an OLE object originally created in another application such as Excel or Visio, and then change the file extension, you can subsequently open the exported file in the original application:

```
[GraphExport]
ImportGraphics=Export
; ExportOleFiles = Yes (makes external .ole files) or No (default)
; when not exported, a .wmf is extracted if possible and kept in DCL
ExportOleFiles=Yes
```

When `ExportOleFiles=Yes`, for each OLE object in your document, **Mif2Go** creates a file with extension `.ole` and places the file in the project directory.

Note: Use this setting only to retrieve OLE objects for use in the application that created them; the retrieved objects do not become usable external graphics files.

To change the file extension of all exported OLE objects to that required by (for example) Visio:

```
[GraphSuffix]
ole=vsd
```

See §31.3.1.2 [Substituting graphics files for HTML](#) on page 888.

31.2.5 Converting graphics with FrameMaker export filters

Mif2Go can use FrameMaker export filters to convert any of the following:

- referenced or embedded graphics
- FrameMaker equations
- FrameMaker vector graphics.

In this section:

§31.2.5.1 [Understanding when FrameMaker export filters are required](#) on page 883

§31.2.5.2 [Understanding FrameMaker export filter limitations](#) on page 883

§31.2.5.3 [Directing Mif2Go to use FrameMaker export filters](#) on page 884

§31.2.5.4 [Controlling graphic size](#) on page 884

§31.2.5.5 [Specifying graphic output format and DPI](#) on page 884

§31.2.5.6 [Specifying a naming convention for converted graphics](#) on page 885

§31.2.5.7 [Converting graphics on reference pages](#) on page 885

§31.2.5.8 [Converting graphics on master pages](#) on page 885

§31.2.5.9 [Converting unanchored graphics on body pages](#) on page 886

See also:

§5.7.2.2.1 [Understanding when to use FrameMaker export filters](#) on page 129

§5.7.2.2.2 [Understanding FrameMaker filter limitations](#) on page 129

31.2.5.1 Understanding when FrameMaker export filters are required

Mif2Go must use FrameMaker export filters to convert the following:

[Compound graphics](#)

[Equations](#)

Compound graphics Unless the images are WMF or BMP *and* you are converting to RTF, **Mif2Go** *must* use FrameMaker export filters to convert compound graphics: referenced or embedded images that include callouts, titles, or second images in the same anchored frame. *The only alternative is to create the compound graphics outside of FrameMaker.*

Equations **Mif2Go** always uses FrameMaker export filters to convert equations. *There is no alternative.* **Mif2Go** can export equations to any of the supported graphics formats.

31.2.5.2 Understanding FrameMaker export filter limitations

On Windows systems, FrameMaker export filters use the GDI image, not the original graphic. The exported graphics are created by GDI rendering, regardless of the original DPI of the graphic. In effect, images are resampled to screen resolution (96 DPI) no matter what DPI you set, which can degrade image quality. Adjusting the DPI at which the resampling is done can help for vector graphics, but not for bitmap graphics. You can get more pixels, but they are interpolated from the 96 DPI rendition.

31.2.5.3 Directing Mif2Go to use FrameMaker export filters

To have **Mif2Go** convert graphics using the FrameMaker export filters, set the following options:

```
[Graphics]
; UseGraphicPreviews = No (default)
; or Yes (use preview bitmaps for frames)
UseGraphicPreviews = Yes

[Setup]
; WriteEquations = No (default)
; or Yes (write only equations as graphics files)
WriteEquations = No
; WriteAllGraphics = No (default)
; or Yes (write all anchored frames as graphics files)
WriteAllGraphics = Yes
```

Or, choose **Write for anchored frames** in the **Mif2Go Export** dialog; see [Table 3-2](#) on page 84.

31.2.5.4 Controlling graphic size

With FrameMaker export filters, **Mif2Go** has to export the whole anchored frame, empty white space and all; the size of a graphic created with FrameMaker export filters is that of the anchored frame in which the graphic is displayed in FrameMaker.

For example, if you include in your document a 2" x 3" graphic enclosed in a 7" x 3" anchored frame (to center it so that there is no runaround); and if you export the graphic using a FrameMaker graphic export filter; the resulting exported graphic is 7" x 3", which is mostly whitespace.

If you are converting to HTML and the graphic in question was imported into FrameMaker by reference, you can get around this problem with the following setting:

```
[Graphics]
UseOriginalGraphicNames=Yes
```

See §31.3.1.2 [Substituting graphics files for HTML](#) on page 888 for more information.

31.2.5.5 Specifying graphic output format and DPI

You can set the output DPI and graphic format for the FrameMaker export filters to use. FrameMaker DPI for a bitmap graphic is not the native DPI of the image; instead it is a measure of rescaling to whatever size you want the image to appear in your FrameMaker document. Generally you want to retain that scaling.

```
[Setup]
; EquationExportDPI = number
; (from 50 to 1200, default 140 or HTML, 120 for RTF)
EquationExportDPI=140
; GraphicExportDPI = number (from 50 to 1200, default 96)
GraphicExportDPI=96
; GraphicExportFormat = BMP, TIFF, WMF (RTF default),
; JPEG (HTML default), GIF, PNG, EPS, PICT, CGM, or IGES
GraphicExportFormat=JPEG
```

If print quality is important, and you have scaled down a screenshot (for example) for on-line help, leave all the original pixels in; the view on screen will be the same, and the printout will be more legible because the printer has more pixels available for the image.

For HTML output, changing the `GraphicExportDPI` setting affects a displayed HTML page only if browser scaling is turned off for the images involved.

31.2.5.6 Specifying a naming convention for converted graphics

The graphics produced by the FrameMaker export filters are named by the FileID followed by the ObjectID in hexadecimal; see §5.3 [Identifying files and objects](#) on page 117 for information about FileIDs and ObjectIDs. You can set the way the name is created:

```
[Setup]
; UseGraphicFileID = Yes (default) or No (single-file projects only)
UseGraphicFileID=Yes
; GraphicNameDigits = 6, or 4 to 8 (for longer or shorter names)
GraphicNameDigits=6
```

31.2.5.7 Converting graphics on reference pages

By default, when you use the FrameMaker export filters (see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883), **Mif2Go** includes graphics on reference pages.

To exclude reference-page graphics:

```
[Setup]
; WriteRefPageGraphics = Yes (default) or No (exclude ref frames)
WriteRefPageGraphics=Yes
```

A change to this setting takes effect only when you are running **Mif2Go** from within FrameMaker.

For HTML output, see §23.5.4 [Converting reference-page graphics for HTML](#) on page 712.

31.2.5.8 Converting graphics on master pages

By default, when you use the FrameMaker export filters (see §31.2.5 [Converting graphics with FrameMaker export filters](#) on page 883), **Mif2Go** excludes graphics on master pages.

To include graphics on master pages:

```
[Setup]
; WriteMasterPageGraphics = No (default) or Yes (write h/f graphics)
WriteMasterPageGraphics=No
```

A change to this setting takes effect only when you are running **Mif2Go** from within FrameMaker.

Export master-page graphics

You can export master-page graphics by specifying both of the following settings (see §5.7.3.1 [Processing all graphics first](#) on page 132):

```
[Setup]
WriteMasterPageGraphics=Yes
GraphicsFirst=Yes
```

For HTML output, the resulting graphics are not referenced in the output file, unless you include them with a macro.

For RTF output, **Mif2Go** tries to assign unanchored images and graphic frames on master pages to a header or footer, whichever is closer.

Note: **Mif2Go** does not currently support unanchored arcs, bezier curves (smoothed polylines), text lines (as opposed to text frames), or arrows that are drawn directly on master pages.

31.2.5.9 Converting unanchored graphics on body pages

For HTML output, unanchored frames on body pages are included by default. To exclude them, set the following option:

```
[HTMLOptions]
; ReAnchorFrames = Yes (default, anchor unanchored frames to first
; para on page) or No (skip unanchored frames)
ReAnchorFrames=No
```

For RTF output, **Mif2Go** always ignores unanchored frames on body pages.

31.2.6 Embedding bitmap graphics in WMF for WinHelp

When you produce files for WinHelp use, **Mif2Go** replaces each of your bitmap graphics with a reference to a .wmf file: an external metafile named after the .rtf file. **Mif2Go** creates these metafiles, and embeds the bitmaps in them, so that the graphics are scaled correctly in the WinHelp file. If it did not, all the bitmaps would be rendered by WinHelp at screen resolution (typically 96 DPI) without regard for their original scale. It usually means the graphics appear at two to four times their original size; this is generally not what you want.

However, if you really do not want **Mif2Go** to wrap the bitmaps in metafiles, the following setting makes **Mif2Go** use them as .bmp files, unscaled:

```
[Graphics]
; EmbedBMPsInWMFs = Yes (default, includes scaling info) or No
EmbedBMPsInWMFs=No
```

The same is true of .wmf files imported into your FrameMaker document. When you choose **File > Save Using Mif2Go...**, **Mif2Go** reads in any external .bmp and .wmf graphics, adds the FrameMaker elements such as callouts, and puts the results out in scaled .wmf files, which are referenced in the .rtf files produced.

Mif2Go's default is to wrap .wmf graphics in new .wmf files. If you do not want them handled this way, set [Graphics] EmbedWMFsInWMFs=No to use them at their native size. There is a risk when **Mif2Go** tries to embed .wmf graphics produced in other applications that the result will look very strange; in that case too, turn off the embedding.

You can specify that all Word frames are to be wrapped:

```
[Graphics]
; WrapAllFrames = No (default) or Yes (to eliminate use of nowrap)
WrapAllFrames=Yes
```

31.2.7 Exporting embedded graphics imported from Word

When you import Word documents that contain embedded images into FrameMaker, you might not have access to the original graphics, and you might not know what tool was used to create them. Even so, you can have **Mif2Go** export those images from FrameMaker in their original formats, and save them as external graphics files.

Internally, Word uses WMF, a vector format, to hold graphics. So you might be getting the Word graphics as WMF, or as something else, depending on the import filter. Internally, FrameMaker stores the embedded images as graphic insets. While the graphics do not have their original file names, they do retain the original data, in FrameMaker's own (lossless) encoding. **Mif2Go** can extract the images and save them in their original formats, with new names. You do not have to know the formats in advance; just tell **Mif2Go** to export everything, including OLE objects, and see what you get. See §31.2.3 [Exporting and converting embedded graphics](#) on page 877.

To replace the embedded graphics in FrameMaker with references to their exported counterparts, see §2.5.3 [Replacing embedded graphics with referenced graphics](#) on page 69.

31.3 Replacing and relocating graphics files

You might want to replace one or more referenced or exported graphics with others that are in a more appropriate format. If you have created an alternate set of graphics files, you might need to direct **Mif2Go** to look for them in a directory different from the location referenced in FrameMaker.

Changing referenced file names and locations for **Mif2Go** requires different settings, depending on whether you are converting to RTF (Word or WinHelp) or to HTML.

This section discusses the following topics:

§31.3.1 [Changing graphics files for HTML output](#) on page 887

§31.3.2 [Changing graphics files for RTF output](#) on page 890

31.3.1 Changing graphics files for HTML output

In this section:

§31.3.1.1 [Specifying graphics location for HTML](#) on page 887

§31.3.1.2 [Substituting graphics files for HTML](#) on page 888

§31.3.1.3 [Overriding path specifications for referenced graphics](#) on page 888

§31.3.1.4 [Using original files and image sizes for referenced graphics](#) on page 889

§31.3.1.5 [Including referenced graphics without converting](#) on page 889

See also:

§23 [Including graphics in HTML](#) on page 703

31.3.1.1 Specifying graphics location for HTML

Graphics files for HTML usually should be in the same directory as the HTML files, or in a related directory. Their location relative to the HTML files might not be the same as their location relative to FrameMaker files. Therefore, you must specify where they will be when your HTML output is deployed on a Web server, in a Help system, or on a production system different from your conversion system.

Note: Some HTML output types restrict placement of graphics; see §23.3 [Locating graphics files for HTML](#) on page 704.

*Graphics in
directory with
HTML files*

To remove any path information from graphics file names, so that a browser or Help viewer will look for graphics in the same directory as the HTML files that reference those graphics:

```
[Graphics]
; StripGraphPath = No (default)
; or Yes (remove path from graphics references)
StripGraphPath = Yes
```

*Graphics in a
different directory*

To specify where a browser or Help viewer should look for graphics:

```
[Graphics]
; GraphPath = path to use (replacing any previous) for all graphics
GraphPath = relative/path/to/graphics/files
```

The location specified by GraphPath is relative to the wrap directory.

*Move graphics to
the referenced
directory*

To move graphics to the specified directory, do one of the following:

- Have **Mif2Go** copy the graphics during conversion; see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.
- Use a system command to copy the graphics; see §34.4 [Executing operating-system commands](#) on page 937.
- Copy the graphics yourself, outside of the conversion process.

See also:

§35.7.4 [Synchronizing graphics settings for HTML output](#) on page 968

§23.3 [Locating graphics files for HTML](#) on page 704

§9.3.10 [Locating graphics files for HTML Help](#) on page 302

§10.3.9 [Getting OmniHelp supporting files in the right place](#) on page 349

§11.3.7.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 380

31.3.1.2 Substituting graphics files for HTML

You can tell **Mif2Go** to use a specific named graphic in place of the original or a generated graphic. For example:

```
[GraphFiles]
; GraphicID (with or without extension) = new name (with extension)
; new name overrides any [Graphics]GraphPath specified
ch01f853.gif = tuner.gif
```

If your FrameMaker document references graphics in non-Web formats (such as TIFF) and you plan to replace those graphics with matching Web-usable images in the same directory, you can specify a new extension for the replacement files. For example:

```
[Graphics]
; GraphSuffix = suffix to use for replacement graphics
GraphSuffix = jpg
```

If some referenced graphics are in a different format (for example GIF), specify the exceptions. For example:

```
[GraphSuffix]
; old suffix = new suffix, overrides [Graphics]GraphSuffix
; jpg = jpg leaves all .jpgs alone even if GraphSuffix=gif
; wmf = png .wmfs are being made into .pngs using a third-party tool
gif = gif
```

31.3.1.3 Overriding path specifications for referenced graphics

To override path settings in [GraphFiles] and in configuration markers (see §33.2.9.4 [Overriding graphic properties for HTML](#) on page 929):

```
[Graphics]
; GraphPathOverrides = No (default) or Yes (overrides any path
; in Config markers and in [GraphFiles], adding GraphPath
; and using FixGraphSpaces)
GraphPathOverrides=Yes
```

When GraphPathOverrides=Yes, **Mif2Go** uses the path to graphics specified by GraphPath (see §23.3 [Locating graphics files for HTML](#) on page 704) instead of any path (or lack of a path) specified in [GraphFiles] (see §31.3.1.2 [Substituting graphics files for HTML](#) on page 888) or in a ***Config** marker that has content:

```
[GraphFiles]=filename
```

Also, **Mif2Go** replaces with underscores any spaces in file names of referenced graphics; see §31.3.1.4 [Using original files and image sizes for referenced graphics](#) on page 889.

31.3.1.4 Using original files and image sizes for referenced graphics

You can have **Mif2Go** use the original referenced graphics files, instead of the files generated by the FrameMaker export filters; and you can also eliminate spaces from the names of those original files, to make the names valid in all environments:

```
[Graphics]
; UseOriginalGraphicNames = No (default, always use previews) or Yes
UseOriginalGraphicNames = Yes
; FixGraphSpaces = Yes (default, replace space with underscore) or No
FixGraphSpaces = Yes
```

When `UseOriginalGraphicNames=Yes`, **Mif2Go** overrides your request to use FrameMaker-exported graphics *only when possible*, which is when a graphic is referenced and alone in its anchored frame.

If you find that some of your referenced graphics are still being exported when `UseOriginalGraphicNames=Yes`, look for artifacts in their anchored frames. In FrameMaker, select an anchored frame, then on the right-click context menu choose **Select all in frame** to see what turns up. For example, if an anchored frame contains two .jpg images, includes callouts, or has a border drawn with FrameMaker drawing tools, **Mif2Go** generates a graphics file using FrameMaker export filters. Also look for empty Text Lines, and objects outside the boundary of the anchored frame; such objects are easy to spot in a MIF representation of the document file, and can be deleted in MIF.

The `` tag references the original name of the graphics file (possibly modified for path and extension by several other settings acting in concert), and uses the FrameMaker-determined size of the original imported image as opposed to the size of the enclosing anchored frame (to preserve scale, also possibly modified by other size settings).

Note: When you set `UseOriginalGraphicNames=Yes`, you are promising that there are graphics files (**Mif2Go** *does not check!*) to be loaded by the resulting .htm using whatever you set for the graphics path and extension; see §31.3.1.5 [Including referenced graphics without converting](#) on page 889.

The size of each graphic will be the size of the actual image, instead of the size of the anchored frame in which it was displayed in FrameMaker. For example, if you put a 2" x 3" graphic in a 7" x 3" frame to center it so that there is no runaround; and you export the graphic using FrameMaker graphic export filters; the resulting exported graphic is 7" x 3", and mostly whitespace. When you use the original graphic, you do not want it stretched to 7" x 3", you want it to be 2" x 3".

31.3.1.5 Including referenced graphics without converting

If the graphics in your FrameMaker document were imported by reference, and they are already in a format and of a size appropriate for HTML output, you can have **Mif2Go** include them *as is*, without passing them through any conversion process.

To use original referenced graphics, in the **Mif2Go Export** dialog check **Use original graphic names**; or, specify the following option in the configuration file:

```
[Graphics]
UseOriginalGraphicNames = Yes
```

Also comment out the following setting, if present:

```
[Graphics]
; GraphSuffix = gif
```

Additional settings depend on where you want graphics placed for output:

[Graphics in the same directory as HTML output files](#)

Graphics in a different directory

Graphics in the same directory as HTML output files

To locate graphics in the same directory as the .htm files, set StripGraphPath=Yes and comment out GraphPath:

```
[Graphics]
StripGraphPath=Yes
; GraphPath = relative/path/to/graphics/files
```

Graphics in a different directory

To locate graphics somewhere other than the directory with the .htm files, use GraphPath to specify the relative path to their location:

```
[Graphics]
GraphPath = relative/path/to/graphics/files
```

The location specified by GraphPath is relative to the wrap directory. See §31.3.1.1 [Specifying graphics location for HTML](#) on page 887.

31.3.2 Changing graphics files for RTF output

For print RTF or WinHelp output, you can direct **Mif2Go** to use graphics files different from those referenced in FrameMaker, or exclude graphics altogether.

In this section:

- §31.3.2.1 [Substituting graphics files for RTF](#) on page 890
- §31.3.2.2 [Using already converted graphics for RTF](#) on page 893
- §31.3.2.3 [Using different bitmaps for print RTF and for WinHelp](#) on page 894
- §31.3.2.4 [Replacing embedded graphics for RTF](#) on page 894
- §31.3.2.5 [Excluding graphics from RTF output](#) on page 895

31.3.2.1 Substituting graphics files for RTF

If any graphics referenced in your FrameMaker document are not BMP or WMF, unless you map those graphics to replacements, **Mif2Go** puts them in an INCLUDEPICTURE field for Word output, and omits them from WinHelp output.

In this section:

- §31.3.2.1.1 [Replacing WMF files with BMP files](#) on page 890
- §31.3.2.1.2 [Substituting files with different extensions](#) on page 891
- §31.3.2.1.3 [Substituting files with different names or locations](#) on page 891
- §31.3.2.1.4 [Understanding replacement examples](#) on page 892

31.3.2.1.1 Replacing WMF files with BMP files

If all you want to do is change the file extension for referenced WMF files that you are replacing with referenced BMP files, and if both of the following are true:

- the BMP files are in the same directory as the WMF files
- none of the BMP files form parts of FrameMaker vector graphics

you can use the following settings:

```
[Graphics]
; NameWMFsAsBMPs = No (default)
; or Yes (to change .wmf refs in the .rtf)
; EmbedBMPsInWMFs = Yes (default, includes scaling info) or No
NameWMFsAsBMPs=Yes
EmbedBMPsInWMFs=No
```

This setting is intended mainly for those who have to replace the WMFs generated by **Mif2Go**, in order to deal with a resource-leak defect in Windows 9x systems. See §8.6.2 [Avoiding the GDI resource leak](#) on page 264.

31.3.2.1.2 Substituting files with different extensions

The simplest way to substitute graphics in a different format is as follows:

- give all the replacement graphics the same base names as the originals
- put all the replacement graphics in the project directory.

Then you can simply map the old extension to the new extension, as follows:

```
[Graphics]
FileNames=Map
FilePaths=None

[GraphFiles]
oldext=newext
```

Do not include a leading dot when you map extensions. For example:

```
[GraphFiles]
jpg=bmp
```

However, if some of your replacement graphics have different base names or are in other directories, mapping old to new files becomes more complex. See §31.3.2.1.3 [Substituting files with different names or locations](#) on page 891.

31.3.2.1.3 Substituting files with different names or locations

You can tell **Mif2Go** to look for replacement files that differ from the original files in any or all of the following respects:

- different location (file path)
- different base file name
- different file extension.

To map referenced graphics to replacements:

```
[Graphics]
; FileNames = Retain (default) or Map (in the GraphFiles section)
; FilePaths (for graphics) = Retain (default) or None (strip off)
FileNames=Map
```

To specify different file paths, different names, or different extensions, when

FileNames=Map:

```
[GraphFiles]
; types to map, replace extension, old=new for referenced graphics
; specific filenames to replace, old = new, overrides type setting
```

Note: When you specify paths in [GraphFiles], use forward slashes for separators.

[Table 31-1](#) shows where **Mif2Go** expects to find replacement files for various combinations of FileNames and FilePaths values and [GraphFiles] settings.

Table 31-1 RTF replacement graphics file mappings and locations

FileNames	Valid [GraphFiles] mappings		Replacement directory when FilePaths =	
	Original graphics file(s)	= Replacement file(s)	Retain	None
Retain	<i>Ignored</i>		Original	Output
Map	ext	= ext	Original	Output
	filename.ext	= filename.ext	Output	Output
	path/filename.ext	= filename.ext	Output	Output
	filename.ext	= path/filename.ext	Per [GraphFiles] path	
	path/filename.ext	= path/filename.ext	Per [GraphFiles] path	

<i>FileNames=Map</i>	When FileNames=Map, Mif2Go uses the settings in [GraphFiles] to find replacements.
<i>FileNames=Retain</i>	When FileNames=Retain, Mif2Go ignores the settings in [GraphFiles], and looks for replacements in one of two places (determined by the FilePaths setting): the same directory as the original graphics, or the project directory.
<i>FilePaths=None</i>	<p>When FilePaths=None, Mif2Go ignores the path component of the file references in FrameMaker. Unless you specify FileNames=Map and a different path in [GraphFiles], Mif2Go looks for replacements <i>only in the project directory</i>.</p> <p>You can use FilePaths=None when you are converting on a system different from the system used for authoring or editing, to avoid replicating the directory structure. This setting prevents problems with attempted access to drives (such as network drives) that do not exist on the system used for conversions, but do exist on the systems used for authoring or editing.</p>
<i>FilePaths=Retain</i>	When FilePaths=Retain, unless you specify FileNames=Map and different paths for both original and replacement files in [GraphFiles], Mif2Go looks for replacement graphics <i>only in the same directory as the original files</i> .
<i>Avoid specifying original file paths</i>	It is best to use FilePaths=None, and put the replacements in the project directory. This is because specifying original file paths in [GraphFiles] is problematic; success depends on exactly matching the paths in FrameMaker, whether they are absolute or relative.

31.3.2.1.4 Understanding replacement examples

If replacement graphics have the same base names as the originals, and are located in the same directory with the originals, and you are replacing some or all GIFs with BMPs:

```
[Graphics]
FilePaths=Retain
FileNames=Map

[GraphFiles]
gif=bmp
```

If all replacement graphics are in the project directory, and you are replacing GIFs with BMPs, and in one instance replacing an existing BMP with a new one:

```
[Graphics]
FilePaths=None
FileNames=Map
```

```
[GraphFiles]
gif=bmp
oldpic.bmp=newpic.bmp
```

If some replacement graphics are *not* with originals and *not* in the project directory, you must specify paths to the replacement files:

```
[Graphics]
FilePaths=None
FileNames=Map

[GraphFiles]
oldpic.bmp=D:/Graphics/Beta/newpic.bmp
```

If some of the replacement graphics are in the same directory as the original graphics, but some of the base file names are different, you must specify both the original and the replacement path:

```
[Graphics]
FilePaths=Retain
FileNames=Map

[GraphFiles]
D:/Graphics/oldpic.bmp=D:/Graphics/newpic.bmp
```

Because path references in FrameMaker could be relative or absolute, it is better to avoid specifying paths to the left of the equals sign in [GraphFiles]; instead, move or copy the replacement graphics to the project directory, and set FilePaths=None.

31.3.2.2 Using already converted graphics for RTF

To instruct **Mif2Go** to use graphics you have already converted to another format, do the following:

1. Make sure each converted graphic has the same name as the graphic it replaces, except for the file extension.

For example, if the original graphics were named:

```
screen01.tif
screen02.tif
screen03.tif
```

and you converted them to WMF format, name the WMF replacements:

```
screen01.wmf
screen02.wmf
screen03.wmf
```

2. Put the converted graphics in one of these directories:
 - the same directory as the original graphics
 - the project directory with the RTF files.
3. Specify settings in your project configuration file, `m2rtf.ini`.
 - 3.1. Specify file-name treatment and replacement-file location:

```
[Graphics]
FileNames=Map
```

If you put the converted graphics in the same directory as the original graphics:

```
FilePaths=Retain
```

If you put the converted files in the project directory with the RTF files:

```
FilePaths=None
```

- 3.2. Map the original file extension to the new file extension; for example:

```
[GraphFiles]
tif=bmp      (if you converted TIFF graphics to BMP format)
eps=wmf      (if you converted EPS graphics to WMF format)
```

31.3.2.3 Using different bitmaps for print RTF and for WinHelp

Mif2Go allows you to use different versions of bitmaps for RTF print documents and for WinHelp. You can do this in one step, if you imported the bitmaps into your FrameMaker document by reference. Otherwise you can have **Mif2Go** export the bitmaps in the first stage of conversion, then you can replace them in the second stage, as described in §31.2.3 [Exporting and converting embedded graphics](#) on page 877. In either case, the bitmaps you prepare for WinHelp are placed in the project directory, rather than where the print-version graphics are located.

31.3.2.4 Replacing embedded graphics for RTF

Suppose you have a FrameMaker document into which someone imported EPS graphics by copying instead of by reference. You want to produce a Word RTF file that references those graphics, and also produce the graphics files themselves, with their original names. **Mif2Go** can do most of that for you, except provide the original file names.

When FrameMaker imports by copying, the original file name is lost; therefore **Mif2Go** cannot access that name. Instead, **Mif2Go** generates a name that consists of the first four characters of the FrameMaker file name, followed by an incremental number, starting with 0001; see §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134.

For example, if your FrameMaker file is `myfile.fm`, default configuration settings would yield `myfi0001.eps`, `myfi0002.eps`, and so forth. You can use the **Mif2Go** Conversion Designer Import Graphics panel to set the number of letters and digits to use in the names, in **Name exported files....**

By default, **Mif2Go** exports EPS files from your FrameMaker document (see §31.2.3 [Exporting and converting embedded graphics](#) on page 877), so if you simply run the conversion you will get external EPS files, and they will be referenced in the resulting Word RTF by their **Mif2Go**-generated names. After you run the conversion and it produces external EPS files, You might want to rename these files to something sensible, and then re-import them *by reference* into your FrameMaker document, replacing the copied-in graphics:

1. In FrameMaker:
 - 1.1. Select a copied-in image (the imported object, not the frame).
 - 1.2. Choose **File > Import** from the main FrameMaker menu.
 - 1.3. Select the renamed replacement graphic file.
 - 1.4. Import that file by reference.
2. In the **Mif2Go** Conversion Designer Import Graphics panel:
 - 2.1. Set **EPSI Usage** to **Both**.
 - 2.2. Click **Apply**
 - 2.3. Click **Update All**.
3. Run the conversion again.

When you first open the resulting RTF file in Word, all referenced graphics files must be in the same directory as the RTF file. After that, the graphics files are no longer needed; when you save in Word format, all graphics are embedded. Even though this is by reference, Word keeps the name around (unlike FrameMaker), and also keeps a full copy of the graphic inside the `.doc` file. Therefore you can expect really large `.doc` files.

31.3.2.5 Excluding graphics from RTF output

To strip all graphics from the files you are converting to RTF, for either Word or WinHelp:

```
[Graphics]
; RemoveGraphics = No (default) or Yes (strip all graphics from doc)
RemoveGraphics=Yes
```

You can keep empty frames, remove them, or identify them by having **Mif2Go** write the name of the missing graphic visibly in the empty frame. For example, to display in RTF output only file names and not the graphics themselves:

```
[WordOptions] or [HelpOptions]
; EmptyFrames = Standard (retain), Remove, or Identify (missing file)
EmptyFrames=Identify
```

31.4 Specifying custom settings for individual graphics

Many of the graphics settings you can specify in the configuration file apply to all the graphics in your document. However, if the right setting for one graphic is wrong for another, you might be able to override the configuration value in your FrameMaker document for an individual graphic.

In this section:

§31.4.1 [Overriding graphics settings with custom markers](#) on page 895

§31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896

See also:

§23.7 [Specifying HTML image attributes](#) on page 718

§25.2 [Applying WAI markup to images](#) on page 756.

§29.2.4 [Using attribute markers for HTML or XML](#) on page 835

31.4.1 Overriding graphics settings with custom markers

You can use custom FrameMaker markers (see §29.2 [Adding custom marker types](#) on page 832) markers to insert configuration overrides in your FrameMaker document. For RTF output, use a **Config** or **RTFConfig** marker for this purpose; for HTML output, a **Config** or **HTMConfig** marker. See §33.2.2 [Overriding settings with configuration markers](#) on page 921. The following tables show which settings can be overridden:

Table 33-2 [Fixed-key configuration sections subject to overrides](#) on page 925

Table 33-6 [HTML graphic sections subject to overrides](#) on page 930

Fixed-key overrides persist until the end of the file, or until changed by another override; variable-key overrides apply only to the next graphic (see §33.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 923).

To override a setting for a given graphic, place the marker somewhere in the text before the graphic-frame anchor and after the anchor of the preceding graphic frame. For marker text, supply the setting you want to change. For example, for RTF output:

```
[Graphics]BitmapDPI=72
```

You can change only one setting per marker; however, you can use as many such markers as you need.

Note: If you change the same setting in the FrameMaker *Object Attributes* dialog, the value in the marker takes precedence.

See also:

§29.1 [Using custom FrameMaker markers](#) on page 831

§33.2 [Overriding settings with markers or macros](#) on page 920

§31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896

31.4.2 Overriding graphics settings with FrameMaker object attributes

For images in anchored frames, in FrameMaker 7.0 and later versions you can assign image attributes via the FrameMaker *Object Attributes* dialog. Using the *Object Attributes* dialog keeps image properties and attributes that apply to a single image contained, so they always accompany the image when you move or copy its anchored frame to another part of your document, or to another document.

Note: If you insert the same setting with a marker just before the anchored frame, the value in the marker takes precedence; see §31.4.1 [Overriding graphics settings with custom markers](#) on page 895.

To assign an attribute to a graphic object in an anchored frame, select the frame and choose **Object Properties...** from the right-click context menu or the FrameMaker **Graphics** menu. In the *Object Properties* dialog, click **Object Attributes...** to open the FrameMaker *Object Attributes* dialog, shown in [Figure 31-1](#) on page 898. Here you can specify attributes for the graphic object in the frame.

*Text Attributes
section*

For HTML output, **Mif2Go** treats whatever you type in the **Text Attributes** section of the *Object Attributes* dialog as follows:

Alternate: HTML tag alt attribute

Actual: HTML tag longdesc attribute

For DITA output, **Mif2Go** treats whatever you type in the **Text Attributes** section of the *Object Attributes* dialog as follows:

Alternate: DITA <alt> tag

Actual: Ignored

See §15.7.4 [Providing alternate text for images](#) on page 518.

*New or Changed
Attribute names*

In the **New or Changed Attribute** section of the *Object Attributes* dialog, **Mif2Go** recognizes and acts on the following **Name** values:

Graph*	Any custom marker name that begins with Graph and ends with the name of a valid HTML attribute.
Config	Any configuration-override marker name that begins with HTMConfig , RTFConfig , or Config , and ends with any additional characters.
GraphGroup	For HTML output, the [GraphGroup] configuration section name.

New or Changed
Attribute
definitions

For **Definition**, supply any value that would be valid in the corresponding marker or configuration section:

Graph*	<p>A value for Mif2Go to assign to the named HTML <code></code> attribute. For example:</p> <p>Name: GraphLowsrc</p> <p>Definition: lowres.jpg</p> <p>causes Mif2Go to include the following attribute in the <code></code> tag:</p> <pre></pre> <p>See §29.2.4 Using attribute markers for HTML or XML on page 835.</p>
Config	<p>Any configuration setting that would otherwise appear in one of the [Graph*] sections subject to overrides. For example:</p> <p>Name: HTMConfig</p> <p>Definition: [GraphDpi]=72</p> <p>causes Mif2Go to change the resolution to 72 DPI for the image.</p> <p>See §33.2.9.4 Overriding graphic properties for HTML on page 929.</p>
GraphGroup	<p>The name of the group to which you want the image assigned. For example:</p> <p>Name: GraphGroup</p> <p>Definition: 3x5pics</p> <p>causes Mif2Go to include the image in group 3x5pics.</p> <p>See §23.5.1.4 Creating named groups of graphics on page 710.</p>

Only one
definition per
attribute name

Because the *Object Attributes* dialog does not allow you to add more than one definition for the same attribute name, **Mif2Go** recognizes any name that *starts with* “Config” as **Config**, and similarly for **HTMConfig** and **RTFConfig**.

For example, to specify two different HTML attributes for the same image:

Name: HTMConfigHi
Definition: [GraphHigh]=50
Name: HTMConfigWd
Definition: [GraphWide]=75

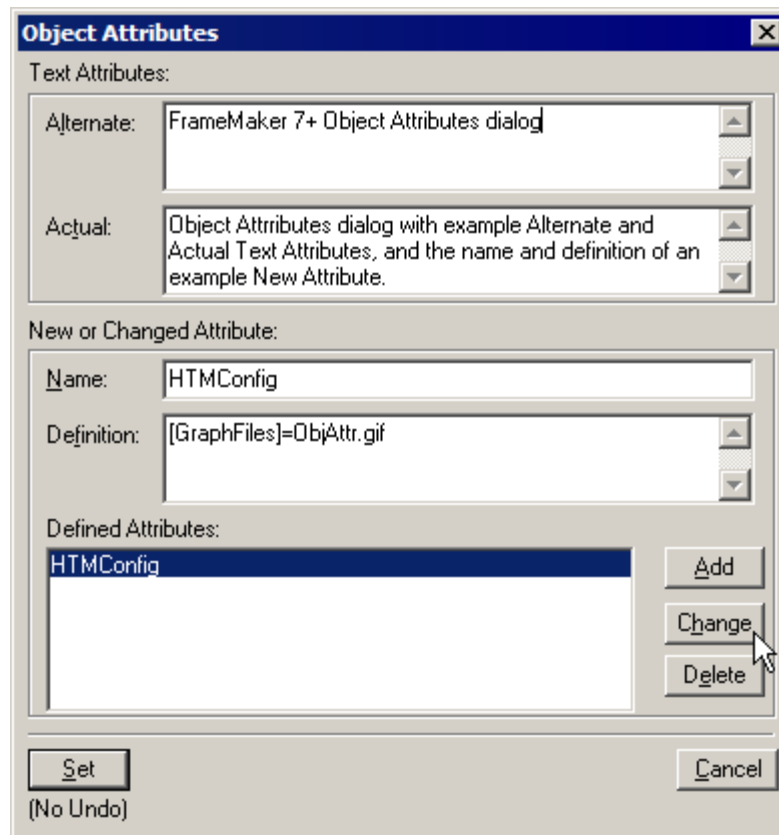
Object Attributes
dialog not always
trustworthy

Be sure to go back and check the settings you add via the *Object Attributes* dialog, because at least in FrameMaker versions 7.0 and 7.1, the dialog seems to be somewhat unstable.

For example, based on our experience at Omni Systems:

- You might have to delete and replace a definition that does not show the correct value when you reopen the dialog.
- If an attribute you add causes FrameMaker to crash the next time you open the dialog, save the file as MIF, then open the MIF file as a regular FrameMaker file again (**Mif2Go** can do this for you; see §D.2.6 [Check for file corruption](#) on page 1032). The offending attribute should be gone, and you can re-enter its name and definition.

Once your entries persist when you reopen the dialog, the values should be reliable.

Figure 31-1 FrameMaker 7+ *Object Attributes* dialog

31.5 Controlling image appearance in RTF output

In this section:

- §31.5.1 [Rescaling bitmap graphics](#) on page 898
- §31.5.2 [Reorienting bitmap graphics](#) on page 899
- §31.5.3 [Compressing bitmap graphics](#) on page 899
- §31.5.4 [Positioning borders around inline graphics](#) on page 900
- §31.5.5 [Mapping FrameMaker pen style patterns](#) on page 900
- §31.5.6 [Converting graphic text](#) on page 901
- §31.5.7 [Specifying transparency for WinHelp 4](#) on page 903

See also:

- §6.14.6 [Positioning graphics and wrapping text](#) on page 191
- §6.14.7 [Preserving graphics scale in Word](#) on page 191
- §8.6.3 [Positioning graphics in WinHelp](#) on page 264

31.5.1 Rescaling bitmap graphics

You can rescale bitmaps using the [Graphics] BitmapDPI setting, either all at once or individually. A setting of BitmapDPI=0 preserves the scaling used in FrameMaker; this is usually best when producing RTF for use in Word.

If you are concerned with screenshots for WinHelp, though, any text in the bitmap is likely to be unreadable if you scale it down at all. Instead, use BitmapDPI=96 to make the screenshot appear at its full original size. If that is too big, you must make a different

version of the bitmap for WinHelp, rescaling it with a graphics program that actually “resamples” it, instead of just dropping rows and columns the way WinHelp does when it rescales.

Normally, **Mif2Go** sets the `StretchMode`, which specifies what method is to be used later when other applications rescale the `.wmf` bitmaps, according to the type of bitmap present. For color bitmaps, **Mif2Go** specifies mode 3, which means to drop the eliminated lines without altering their neighbors. For monochrome, **Mif2Go** specifies mode 2, which preserves black detail by ORing the bits in the adjacent lines; you can also set mode 1, which keeps white detail by ANDing the bits.

```
[Graphics]
; StretchMode for bitmaps =
; 0 (default),
; 1 (black),
; 2 (white),
; 3 (color)
StretchMode=0
```

When **Mif2Go** wraps external WMFs in generated WMFs, so that they are scaled correctly, the fill properties for the objects drawn are determined by the brush in effect. If everything turns black, it may be that the maker of the original WMF assumed a different default brush from the one **Mif2Go** assumes. You can try other possible starting values for the brush (White and Hollow) to see if that improves the appearance of the graphic:

```
[Graphics]
; DefBrushType = Black (default), White, or Hollow
DefBrushType=Black
```

31.5.2 Reorienting bitmap graphics

Sometimes you might find that your bitmaps in the RTF file are upside down, and somehow mirrored. This happens when the bitmap starts with the top scan line instead of the bottom one. Unfortunately, the `.bmp` format does not contain any flag to indicate this usage (which is increasingly common). **Mif2Go** cannot identify such bitmaps; you must set `[Graphics] BitmapFlip=1`. Generally if one bitmap is flipped, all the rest from the same source are too, so the setting can often apply to the whole file. If some are flipped but not others, you will need to add the setting as a marker before every graphic’s anchor in the text, specifying `BitmapFlip=1` to flip or `BitmapFlip=0` to leave it as is.

31.5.3 Compressing bitmap graphics

Some printer drivers have problems with compressed bitmaps, which is the default for **Mif2Go**. Other printer drivers have trouble with *uncompressed* bitmaps. If your bitmap (usually color) looks fine on screen, but prints out black and white (on a color printer), you might want to try changing these settings:

```
[Graphics]
; CompressRasters = Yes (default, compress 16 and 256 color BMPs)
; or No
CompressRasters=Yes
; RasterBorders = No (default), Inside (raster), Outside, or Frame
; Frame centers border on edge, as in FM, but results in narrowing
; to half width if graphic edge is at an edge of its anchored frame
RasterBorders=No
```

31.5.4 Positioning borders around inline graphics

If borders around inline graphics do not look right in Word output, try one of the following settings:

```
[Graphics]
; FrBorders = Frame (default, centered), Inside, Outside, or None
FrBorders=Frame
```

Each of the values for `FrBorders` (except `None`) moves the position of the border rectangle itself slightly. You might not see a difference unless the border is thick, because the move is just half the border thickness.

If you do not get satisfactory results with `FrBorders`, try the following option instead:

```
[Graphics]
; FrameBorders = No (default, no borders on inlined frames) or Yes
; This is best used when such frames are alone in their paragraphs.
; Should not be used unless FrBorders does not operate correctly.
FrameBorders=Yes
```

Both methods work for in-line graphic frames that are alone in their paragraphs. However, they do very different things:

- `FrBorders` adds the border rectangle to the WMF graphic itself before embedding it in the output.
- `FrameBorders` uses Word paragraph properties to set a border for the paragraph that contains the (in-line) frame.

Use one or the other, not both

If you set both `FrBorders` and `FrameBorders`, you might get a thicker border; or if the graphic is not alone in its paragraph, you might get a mess. For in-line graphics positioned “at insertion point”, if you set `FrameBorders=Yes`, and there is other text in the paragraph, the text gets the border along with the frame, all in one box together. Not likely to be what you want. This does not happen when you use `FrBorders`.

If you set neither `FrBorders` nor `FrameBorders`, any border around an in-line graphic is centered on the graphic.

Set line spacing above

To get the correct line spacing at the top of in-line frames, also specify the following setting:

```
[Graphics]
NoBlankFirstGTLine=No
```

See §31.5.6.5 [Fine-tuning graphic text](#) on page 902.

31.5.5 Mapping FrameMaker pen style patterns

FrameMaker’s graphic pen style patterns have no direct correspondence to the available WMF line styles. For RTF output for which **Mif2Go** converts FrameMaker native graphics (without using FrameMaker’s graphic export filters), you can map the conversion. The values shown are the defaults; defaults for 1 through 6 are all zero (solid):

```
[GraphLineStyle]
; FrameMaker pen style number = WMF line style
8=1
9=1
10=2
11=2
12=4
13=3
14=3
```

Valid FrameMaker pen styles are 1-6 and 8-14, excluding 0 (black), 7 (white), and 15 (invisible), which have fixed mappings. Valid WMF line styles are 0 (solid, the default), 1 (dashed), 2 (dotted), 3 (dot-dash), 4 (dash-dot-dot), and 5 (invisible); all but solid force the line width to the minimum (a WMF requirement).

The [GraphLineStyles] section is effective only when FrameMaker graphics are being converted by **Mif2Go**, without using the FrameMaker graphic export filters.

31.5.6 Converting graphic text

A FrameMaker graphic can contain the following kinds of text:

- individual text lines
- text paragraphs in text frames.

Mif2Go normally converts both kinds to WMF text lines, approximating text features that are not available in WMF.

In this section:

- §31.5.6.1 [Matching font and point size](#) on page 901
- §31.5.6.2 [Specifying a default graphics font](#) on page 902
- §31.5.6.3 [Converting graphic text to text](#) on page 902
- §31.5.6.4 [Clipping text outside a text frame](#) on page 902
- §31.5.6.5 [Fine-tuning graphic text](#) on page 902
- §31.5.6.6 [Specifying graphic text background](#) on page 903

31.5.6.1 Matching font and point size

Problems with text size or font in converted graphic text might indicate an unusual FrameMaker default. **Mif2Go** defaults set the font as Times New Roman, and the size at 1200. Check the settings in FrameMaker as follows:

1. Open FrameMaker **Character Designer** (Ctrl-D).
2. Open a new FrameMaker document.
3. Insert an anchored frame in the document.
4. Type a text line in the anchored frame, using the “A” tool on the graphics palette.
5. Click the text you just typed.

If the font and size shown in **Character Designer** are different from the **Mif2Go** defaults, adjust the **Mif2Go** settings to match the FrameMaker settings:

```
[Graphics]
; FrameDefaultFontName = name of default font in FrameMaker graphics
; must match setting Frame uses internally for this purpose
FrameDefaultFontName=Times New Roman
; FrameDefaultFontSize = size of default font in FrameMaker graphics
; must match setting Frame uses internally (hundredths of a point)
FrameDefaultFontSize=1200
```

If graphic text properties (font, size, color, variation, etc.) still do not convert properly, try this setting:

```
[Graphics]
; UseDefaultGraphicFormat = No (default), or Yes for graphics
; text formats, if text properties are not being maintained correctly
UseDefaultGraphicFormat=Yes
```

31.5.6.2 Specifying a default graphics font

If most of your graphics use the same font, at the same size, you can reduce the amount of storage required for graphics in the output by specifying these settings:

```
[Graphics]
; DefFont = name of default font in WMF graphics
DefFont=Arial
; DefFSize = size in twips for default font in WMF graphics
DefFSize=180
```

31.5.6.3 Converting graphic text to text

If a graphic consists of a single text frame, **Mif2Go** can output the text in a Word text frame or as normal WinHelp text instead of in a graphics frame; the rendition is usually much closer to that in FrameMaker, and is more easily edited:

```
[Graphics]
; GraphText = Embed (as for captions), Frame (as text), or Text
; applies only to "graphics" consisting solely of one text frame
GraphText=Embed
```

The default is to embed graphic text, which for Word produces a WMF Word Picture containing the text. Although it is not editable text, a Word Picture can be easier to position correctly than a Word text frame.

WinHelp does not allow text frames. For WinHelp, `GraphText=Frame` is equivalent to `GraphText=Text`, and the text is output as normal text instead of being included in a WMF.

31.5.6.4 Clipping text outside a text frame

Although FrameMaker clips text that goes below the bottom of its text frame, the text is still present in the .mif file. You can specify how such text should be handled:

```
[Graphics]
; ClipType =
; Show (leave alone),
; Move (up into frame, default), or
; Delete (remove the text entirely)
; applies only to graphic text in Frame native vector graphics
ClipType=Move
; ClipLimit = twips to allow below frame before clipping graphic text
ClipLimit=20
```

`ClipLimit` sets the number of twips (twentieths of a point) to allow the text baseline to go below the frame before the `Move` or `Delete` option takes effect. The default is 20 (one point). You might have to adjust the value of `ClipLimit` if either of the following happens:

- `ClipType=Move` and text at the bottom of the frame still disappears: increase `ClipLimit`.
- `ClipType=Delete` and unwanted text is still visible: decrease `ClipLimit`.

Negative values of `ClipLimit` are acceptable.

31.5.6.5 Fine-tuning graphic text

Sometimes the text does not fit in the converted graphic quite the same way it fit in the original. Often this is a result of differing font metrics. Your font mapping choices (see §6.9 [Specifying font usage](#) on page 166) affect graphics text, also.

You can adjust text properties in graphics several ways:

```
[Graphics]
; GrVertAdjust = half-points to adjust in-line graphics down
; (neg for up)
GrVertAdjust=4
; TextScale = value in percent to apply to font sizes in graphics
TextScale=100
; TextWidth = percent of unscaled font height for font widths
; in graphics
TextWidth=0
; TextVertAdjust = twips to move text in metafiles down (neg for up)
TextVertAdjust=0
; NoBlankFirstGTLine = Yes (default,
; ignore blank first line in graphic text frame) or No
NoBlankFirstGTLine=Yes
; UseTopSpaceAbove = No (ignore first space above in callouts) or Yes
UseTopSpaceAbove=No
; SuppressGTUnderlines = No (default) or Yes (no underlines)
SuppressGTUnderlines=No
; FrameExactHeight = 0 (default, auto) or 1 (size as original)
FrameExactHeight=0
```

Change these settings only if the normal defaults yield unacceptable results:

GrVertAdjust	Move in-line graphics up or down by half-point increments
TextScale	Percentage of the original font size; this affects both height and width of characters.
TextWidth	Percentage of character height; 0 leaves it at the normal proportion for the font (usually about 35).
TextVertAdjust	Number of twips (twentieths of a point) to move the text down (or up, using negative numbers).
UseTopSpaceAbove	Keep or ignore the first space above callouts.
NoBlankFirstGTLine	Keep or ignore a blank first line.
SuppressGTUnderlines	Keep or ignore underlines in callouts.
FrameExactHeight	Keep the original height of the frame, or auto-scale it.

31.5.6.6 Specifying graphic text background

You can specify whether the background around graphic text is transparent or opaque:

```
[Graphics]
; BackMode = 1 (transparent) or 2 (opaque) for graphic text
BackMode=1
```

31.5.7 Specifying transparency for WinHelp 4

For WinHelp 4 only, you can choose to make bitmap graphics transparent:

```
[Graphics]
; Transparent makes white bitmap pixels transparent (WinHelp 4 only)
Transparent=Yes
```

31.6 Converting graphics with Microsoft Word filters

As a “last resort” way to convert graphics, you can try using Microsoft Word filters if the usual graphics export procedures do not suffice.

If a graphic imported into or exported from your FrameMaker document is neither WMF or BMP, and you have not mapped that graphic to another that is WMF or BMP, **Mif2Go** inserts an INCLUDEPICTURE field (Word 8) or IMPORT field (Word 7) in the RTF output to give Word a crack at the graphic. That is, **Mif2Go** puts the name of any graphic that it cannot process into an INCLUDEPICTURE (or IMPORT) field in the Word RTF file. If the graphic is in a format Word accepts, Word runs its own filter to import the graphic, creating an internal WMF in the process. The internal WMF is what you see as the graphic rendition in Word.

See §5.7 [Processing graphics](#) on page 126 for normal export options for graphics.

32 Working with content models

Mif2Go provides built-in configurations for content models for DITA and DocBook. This section shows how to modify or replace a built-in content model, or generate a new content model from a valid DITA or DocBook DTD (Document Type Definition). Topics include:

- §32.1 [Understanding Mif2Go content models](#) on page 905
- §32.2 [Modifying or replacing a content model](#) on page 905
- §32.3 [Preparing a content model for use with Mif2Go](#) on page 907
- §32.4 [Understanding content-model configurations](#) on page 908
- §32.5 [Understanding how Mif2Go uses content models](#) on page 911
- §32.6 [Inspecting and correcting element types](#) on page 912
- §32.7 [Specializing or modifying DITA topic types](#) on page 913
- §32.8 [Extracting content-model debug information](#) on page 918

See also:

- §15 [Converting to DITA XML](#) on page 473
- §17 [Converting to DocBook XML](#) on page 557
- §F [Content model configuration](#) on page 1043

32.1 Understanding Mif2Go content models

A **Mif2Go** content model is a configuration-style representation of a DTD. A content-model configuration summarizes DTD information in a form **Mif2Go** can use to produce XML output that conforms to the DTD. **Mif2Go** provides built-in content models for basic DITA version 1.0 and 1.1 topic types, and for DocBook version 4.5. You do not have to include anything special in your **Mif2Go** conversion project to use these content models.

The **Mif2Go** built-in content models were derived from:

- <http://docs.oasis-open.org/dita/v1.0.1/dtd/> for DITA version 1.0
- <http://docs.oasis-open.org/dita/v1.1/CS01/dtd/> for DITA version 1.1
- <http://www.oasis-open.org/docbook/xml/4.5/> for DocBook 4.5.

These content models are complete. You should not need to modify any of them, except possibly to correct element type assignments; see §32.6 [Inspecting and correcting element types](#) on page 912. Each built-in content model has a matching configuration file.

The DTD for DITA version 1.2 is available here:

- <http://docs.oasis-open.org/dita/v1.2/cs01/dtd1.2/>

You can use utility program **dttd2ini** to abstract content models from this and other DTDs; see §32.2.2 [Generating a content model from a DTD](#) on page 906.

32.2 Modifying or replacing a content model

To modify a **Mif2Go** built-in content-model, first locate and extract the appropriate content-model configuration file.

To *replace* a built-in content model, or to add a content model for a new DITA topic type, generate a content-model configuration file from an appropriate DTD.

In this section:

§32.2.1 [Obtaining a copy of a built-in content-model](#) on page 906

§32.2.2 [Generating a content model from a DTD](#) on page 906

32.2.1 Obtaining a copy of a built-in content-model

If you need to modify one of the **Mif2Go** built-in content models to correct an element type assignment (see §32.6 [Inspecting and correcting element types](#) on page 912), you must first extract a configuration file for the content model from the appropriate content-model archive. You can download these archives from the Omni Systems Web site.

Archives of content models are available for the following DTDs:

<u>DTD</u>	<u>Content model archive</u>
DITA version 1.0	dita10contentmods.zip
DITA version 1.1	ditallcontentmods.zip
DocBook version 4.5	docbook45contentmods.zip

Each archive contains both content-model configuration files and the DTD-to-model configuration files used by utility program **dtd2ini** to generate the content models. [Table 32-1](#) lists the configuration files in each archive. Extract from the relevant archive the content-model configuration file you wish to modify.

Table 32-1 Configuration files for Mif2Go built-in content models

Content model archive	Content model configurations	DTD-to-model configurations
dita10contentmods.zip	ditaconcept10.ini	dtd2concept10.ini
	ditamap10.ini	dtd2map10.ini
	ditareference10.ini	dtd2reference10.ini
	ditatask10.ini	dtd2task10.ini
	ditatopic10.ini	dtd2topic10.ini
ditallcontentmods.zip	ditabookmap11.ini	dtd2bookmap11.ini
	ditaconcept11.ini	dtd2concept11.ini
	ditaglossary11.ini	dtd2glossary11.ini
	ditamap11.ini	dtd2map11.ini
	ditareference11.ini	dtd2reference11.ini
	ditatask11.ini	dtd2task11.ini
	ditatopic11.ini	dtd2topic11.ini
docbook45contentmods.zip	docbook45a.ini	docbook45a.ini
	docbook45b.ini	docbook45b.ini

32.2.2 Generating a content model from a DTD

Utility program **dtd2ini** can produce, from a valid DTD, a content-model configuration file to use with **Mif2Go** for DITA or DocBook XML output.

Because **dtd2ini** is GPL software (GNU General Public License), this utility cannot be packaged with any non-GPL software. Therefore **dtd2ini** is not included in your **Mif2Go** distribution. However, you can download **dtd2iniMwin.zip** from the Omni Systems Web site:

<http://mif2go.com>

where *NN* is the **dtd2ini** version number.

To generate a content-model configuration file:

1. Extract the following files from archive `dtd2iniNNwin.zip`:

<code>dtd2ini.exe</code>	(program)
<code>dtd2ini.txt</code>	(instructions)
<code>dtd2ditatopic.ini</code>	(for a DITA specialization), or
<code>dtd2docbook.ini</code>	(for a DocBook DTD).

2. Edit the `dtd2*.ini` file you extracted, and save it as `dtd2ini.ini`.
3. Copy `dtd2ini.exe` to `%OMSYSHOME%\common\bin`.
4. Follow the instructions in `dtd2ini.txt` to produce a content-model configuration file, one of the following:

DITA	<code>DITAtopic_{type}.ini</code> , where <i>topic_{type}</i> is the name of the topic type you are adding or replacing; this is also the name of the content model
DocBook	<code>contentmodel.ini</code> , where <i>contentmodel</i> is any name you choose.

32.3 Preparing a content model for use with Mif2Go

If you plan to use a built-in content model *as is*, you do not need to do anything described in this section.

To prepare a new, modified, or replacement content-model configuration for use with **Mif2Go**:

1. Inspect and (if necessary) change element type assignments; see §32.6 [Inspecting and correcting element types](#) on page 912.
2. **For DITA only**, if you are adding or replacing a content model, provide information needed by **Mif2Go** that is not available in the DTD:
 - Most settings in section `[Topic]` except for `TopicRoot`; see §32.4.1 [Content model \[Topic\] settings](#) on page 909.
 - Table structure information; see §32.7.7 [Providing table structure information for DITA topic types](#) on page 916.

If you generated the content model from a DTD and you plan to rerun **dtd2ini**, also include in configuration file `dtd2ini.ini` as overrides any `[Topic]` and `[*Table]` settings you add. See `dtd2ini.txt` for instructions.

3. Include the following setting in the content-model configuration file:

```
[Topic]
; ModelName = name of type (usually a built-in) to be replaced
; after this file loads, effective only when this file is
; specified in [DITAContentModels] or
; [DocBookOptions]ContentModel in the project configuration file;
; overrides the default use of the filename (without "DITA").
ModelName = contentmodelname
```

`ModelName` specifies either the name of an existing content model to be replaced by the current content model, or a name for the new content model to be added.

If you are replacing a built-in content model, the value for `ModelName` must be one of the following, depending on the output type:

<u>Output type</u>	<u>Built-in content model to be replaced by current model</u>
DITA 1.0	topic, concept, task, reference, or map
DITA 1.1	topic, concept, task, reference, map, glossary, or bookmap
DocBook	book or article

If you assign any other value to `ModelName`, **Mif2Go** adds the new name to the list of models. For example, to add a new DITA topic type widget defined in content-model configuration file `DITAwidget.ini`, in `DITAwidget.ini` you would include the following setting:

```
[Topic]
ModelName = widget
```

To replace the built-in DITA reference content model with a model you have defined in content-model configuration file `DITAmyref.ini`, in `DITAmyref.ini` you would include the following setting:

```
[Topic]
ModelName = reference
```

4. Place the new or modified content-model configuration file in your DITA or DocBook project directory.
5. Specify the base name of the content-model configuration file in your project configuration file:

For DITA, add the base name of each new or modified content-model configuration:

```
[DITAContentModels]
; Topic type name = any text (not used)
DITAtopictype = replaced or new topictype content model
```

For DocBook, specify the base name of the new or modified content-model configuration:

```
[DocBookOptions]
; ContentModel = name of content-model .ini, without extension,
; with which to replace the built-in DocBook 4.5 content model.
ContentModel = otherdocbookmodel
```

32.4 Understanding content-model configurations

A content-model configuration file includes the following sections:

<code>[Topic]</code>	Lists the root element used to generate the content model. Also includes <code>PUBLIC</code> and <code>SYSTEM</code> identifiers for DITA or DocBook, and the starting topic and body element for DITA topic types. See §32.4.1 Content model [Topic] settings on page 909.
<code>[ElementSets]</code>	Groups elements into sets for assignment in sections <code>[TopicParents]</code> and <code>[TopicFirst]</code> . See §32.4.2 Content model [ElementSets] settings on page 910.
<code>[TopicParents]</code>	Lists the valid parent element(s) of each element. See §32.4.3 Content model [TopicParents] settings on page 910.
<code>[TopicFirst]</code>	Lists parent elements for which a given element must be the first child. See §32.4.4 Content model [TopicFirst] settings on page 910.

[TopicLevels]	Specifies required levels for certain elements. See §32.4.5 Content model [TopicLevels] settings on page 911.
[ElementTypes]	Classifies each element as to whether it is block or inline, whether it allows text, and whether it is preformatted. See §32.6 Inspecting and correcting element types on page 912.
[*Table]	<i>DITA only:</i> these sections provide information about table structure that cannot be abstracted from DITA topic-type DTDs. See §32.7.7 Providing table structure information for DITA topic types on page 916.

In this section:

- §32.4.1 [Content model \[Topic\] settings](#) on page 909
- §32.4.2 [Content model \[ElementSets\] settings](#) on page 910
- §32.4.3 [Content model \[TopicParents\] settings](#) on page 910
- §32.4.4 [Content model \[TopicFirst\] settings](#) on page 910
- §32.4.5 [Content model \[TopicLevels\] settings](#) on page 911

32.4.1 Content model [Topic] settings

The following content-model settings specify information for either a DITA or a DocBook content model:

```
[Topic]
; TopicRoot = name of root element for this content model.
TopicRoot = concept
; PrologDType = PUBLIC name used in DOCTYPE header.
PrologDType = "-//OASIS//DTD DITA Concept//EN"
; PrologDTD = SYSTEM name, such as "concept.dtd", can include a path.
PrologDTD = "http://docs.oasis-open.org/dita/v1.1/CD01/dtd/concept.dtd"
; ModelName = name of content model.
ModelName = contentmodelname
```

Root element TopicRoot is the name of the root element for the content model. For DITA, TopicRoot is the name of one of the built-in topic types: topic, concept, task, reference, map, or (for DITA version 1.1) glossary.

Identifiers Double quotes are required for the PUBLIC name and the SYSTEM name. If the SYSTEM name is less than 16 characters long, you *must* prefix the name with two spaces. For example:

```
PrologDTD=  "xyz-topic.dtd"
```

Mif2Go always removes the first space after the equals sign, and retains any subsequent spaces. DOCTYPE styles differ: some require an indent, some prohibit an indent, some want a return, some do not. **Mif2Go** includes a return automatically if (and *only* if) the SYSTEM name is more than 16 characters long. Therefore a shorter SYSTEM name requires a leading space, to separate it from the preceding PUBLIC name when the DOCTYPE header is generated.

Replaced content model If you are providing a replacement content model, ModelName specifies the name of the built-in content model to be replaced by the current content model. ModelName is effective only when the current content model is listed in [DITAContentModels], or specified for [DocBookOptions]ContentModel, in your project configuration file.

DITA-only settings The following settings apply only to DITA content models:

```
[Topic]
; TopicStart = name of element that starts topic, such as "glossterm"
; (for glossary) or "title" (for every other type).
```

```

TopicStart = title
; TopicBody = name for its body element, such as conbody for concept.
TopicBody = conbody
; TopicDerivation = name of type from which it is derived.
TopicDerivation = topictype

```

See §32.7 [Specializing or modifying DITA topic types](#) on page 913.

32.4.2 Content model [ElementSets] settings

To specify groups of elements as the values for certain settings, content-model configuration files define sets of elements:

```

[ElementSets]
; Name for set = list of elements and element sets, separated by
; spaces.
*setname = element1 element2 *otherset

```

Set names start with “*”. Sets can include other sets. Included sets must be defined in [ElementSets] above the sets that include them. Within each set, the elements are alphabetical; that is for convenience in human look-up, and need not be preserved. They *do* have to be one line each; do not use an editor that wraps the lines in [ElementSets].

Each set has an alphanumeric name prefixed with an asterisk. Names of members of the set are listed to the right of the equals sign, separated by spaces. A member of an element set can be either of the following:

- the name of an element
- the name of a previously defined element set.

This allows elements to be grouped for use on the right side of the equals sign in [TopicParents] and [TopicFirst], so that the same set of parents can be used in more than one setting.

Element sets are roughly equivalent to the parameter entities used in DTDs.

32.4.3 Content model [TopicParents] settings

These settings specify the possible parents of each element:

```

[TopicParents]
; element = single parent or single *elementset or Any or No

```

All elements are listed to the left of the equals sign, other than (for DITA) the topic type itself and the topic body type. If an element has more than one possible parent, those parents are defined as a single set, listed in [ElementSets]; see §32.4.2 [Content model \[ElementSets\] settings](#) on page 910.

Each of the items listed to the right of the equals sign is one of the following:

- an element name (single parent)
- an element set name (set of possible parents)
- either of two reserved parent names:

Any	Any parent is acceptable; mainly for inline elements
No	No parent is acceptable; for DITA, this includes elements present in the derived-from type that are excluded from the specialized type.

32.4.4 Content model [TopicFirst] settings

If an element must be the first child of its parents, the element is listed here:

```
[TopicFirst]
; Child element = parents, where child must be the first child of the
; specified parents.
```

If an element must be the first child of more than one possible parent, those parents are defined as a single set, listed in `[ElementSets]`; see §32.4.2 [Content model \[ElementSets\] settings](#) on page 910.

One of the following is assigned to each child element that must be the first child, either of a single parent or of any member of a set of parents:

- an element name (single parent)
- an element set name (set of possible parents)
- either of two reserved parent names:

Any	Must be the first child of every possible parent
No	Must not be the first child of any parent; for DITA, this includes elements present in the derived-from topic type that are excluded from the specialized topic type.

For any child element listed to the left of the equals sign that is *not* the first child of a specified parent, when processing your FrameMaker document **Mif2Go** closes the current parent and opens a new instance of that parent.

Settings in `[TopicFirst]` are used mainly for lists, and for the DITA `<title>` element.

32.4.5 Content model `[TopicLevels]` settings

Each element that must be at a specific level is listed here:

```
[TopicLevels]
; Element name = required level in topic
```

Levels are specified only for elements that must be at a specific level, such as DITA `shortdesc`, `prolog`, `body`, and `related-links` at level 1, and DITA `example` and `metadata` at level 2.

The content models generated by **dtdd2ini** name only level 1 elements in this section.

See also:

§15.5.13 [Specifying DITA element levels](#) on page 509

§17.5.11 [Specifying DocBook element levels](#) on page 579

32.5 Understanding how Mif2Go uses content models

Where there are multiple possible parent elements of a given DITA XML element, a set is defined for those parent elements in the `[ElementSets]` section of the content model configuration file; see §32.1 [Understanding Mif2Go content models](#) on page 905. The `*PartN` sets in this section are computer generated to keep the lengths of the individual sets short enough to be editable; they have no other special purpose. Within each set, elements are listed alphabetically for convenience in human look-up.

For example:

```
[TopicParents]
data=data=*data

[ElementSets]
*data=data-about *Part2 *Part6 *Part9 *Part10
. . .
```



```

*Part2=b cite codeblock codeph data i lq note p ph pre q screen
shortdesc sub sup title tt u xref
. . .
*Part6=abstract dd ddhd desc draft-comment dt dthd entry example fn
itemgroup li lines linkinfo pd pt section sli stentry
. . .
*Part9=alt author brand category copyrholder filepath index-base
index-see index-see-also index-sort-as indexterm msgblock msgph
prodname publisher source systemoutput uicontrol userinput
. . .
*Part10=body component coords delim featnum fig fragref linktext
metadata navtitle oper platform prognum prolog repsep searchtitle sep
series var

```

Mif2Go uses a complex algorithm to determine which element to interpolate in places in your document where a parent element is required. When **Mif2Go** processes your document and encounters text that you have mapped to a `<data>` element (for example), **Mif2Go** searches the above element sets, in sequence, for the current parent element. If the parent is not found, **Mif2Go** performs a graph analysis, breadth-first, of possible parent series that could fit under the current parent. In each case, **Mif2Go** takes the *first* of those candidate parents with equal-length sequences and interpolates it between the `<data>` element and its current parent.

This means that you could change the usage priority of interpolated parents by altering the order of items in a content-model element set. (The full collection of algorithms is rather more complex; for example, **Mif2Go** also considers closing existing parents to find a better solution to the graph problem.)

Suppose you want to tell **Mif2Go** *not* to use certain elements; for example, “forget about `<data>`” or “never use `<fn>` in a `<fig>`”. If you delete `data` from all element sets, this element will never be interpolated into your DITA XML output. If you delete `fig` from all element sets that contain possible parents of `fn`, `fig` will never be interpolated as a parent of `fn`. However, we advise *not* adding or removing any items, because doing so can result in invalid DITA XML. (Removal is safer than addition.)

32.6 Inspecting and correcting element types

Utility program `dttd2ini` cannot always determine from a DTD the correct type of an element. Examine the classifications in section [ElementTypes] of the content-model configuration file, and correct any that are not right.

Element types are as follows:

Block	Block element that does not allow text content
Block Text	Block element that allows text content
Block Text Preform	Block element with preformatted text
Inline	Inline element that does not allow text content
Inline Text	Inline element that allows text content

The default element type is Block without Text.

Block and Inline properties determine whether returns are inserted before start tags and after end tags. The Text property determines whether an attempt is made to wrap any invalid text (in an element that does not allow Text) in a valid container element, such as `<ph>` for DITA. Preform determines whether whitespace within the element is retained *as is*. Preform elements are always Block elements, and they always allow Text.

If you generated the content model from a DTD and you plan to rerun `dttd2ini`, include any changed `[ElementTypes]` settings as overrides in configuration file `dttd2ini.ini`. You can override the `Block`, `Inline`, and `Preform` properties, but not the `Text` property.

Getting the `Block` vs. `Inline` typing wrong for an element is not a major disaster. The element type primarily affects the way XML output is formatted. Most XML processors ignore the formatting, except for preformatted elements.

DITA only: If your DTD defines a block element with no text (for example, to include just markers in the paragraphs from which the element is mapped), also map the no-text block element to `No` in `[DITAParaTags]`, in your configuration file; see §15.4.3.1 [Assigning DITA elements to FrameMaker paragraph formats](#) on page 487. That way you will not be forced to use `CodeBefore` and `CodeAfter` settings to insert the tags for such an element.

32.7 Specializing or modifying DITA topic types

To include custom specialized topic types or maps in your DITA project, you must provide a separate content-model configuration file for each new topic type or modified map or bookmap DTD. You can derive a new type from any of the built-in topic types `topic`, `concept`, `task`, `reference`, `map`, or `glossary` (DITA 1.1 only), or from another specialized type for which you provide a DTD.

To produce the constraints supported by DITA 1.2, you can run utility program `dttd2ini` (see §32.2.2 [Generating a content model from a DTD](#) on page 906) on a local document type shell, and reference the result in your project configuration chain.

In this section:

§32.7.1 [Creating a content model for a specialized topic type](#) on page 913

§32.7.2 [Overriding settings in a DITA content model](#) on page 914

§32.7.4 [Overriding declarations in a DITA map content model](#) on page 915

§32.7.5 [Listing DITA topic type configuration files](#) on page 915

§32.7.6 [Locating DITA topic type configuration files](#) on page 916

§32.7.7 [Providing table structure information for DITA topic types](#) on page 916

32.7.1 Creating a content model for a specialized topic type

To create a content model for a specialized DITA topic type:

1. Run utility program `dttd2ini` with the DTD file for your specialized type as input. Specify for output a content-model configuration file with a name of the form `DITAnewtype.ini`, where *newtype* is the name of the new topic type you are defining. See §32.2.2 [Generating a content model from a DTD](#) on page 906.
2. Add the following settings to `DITAnewtype.ini`:

```
[Topic]
; TopicStart = name of element that starts topic, such as
; "glossterm" (for glossary) or "title" (for every other type).
TopicStart = title
; TopicBody = name for its body element, such as conbody for
; concept.
TopicBody = conbody
```

The required starting element is `<title>` for all built-in DITA topic types (including `map`), except for `glossary`. For `glossary` topics, the starting element is `<glossterm>`. For a specialized topic type, your DTD specifies the starting element.

When the FrameMaker format mapped to the `TopicStart` element in `[DITAParaTags]` is also mapped to level 1 in `[DITALevels]`, that format always starts a new topic of the specialized type. See §15.5.13 [Specifying DITA element levels](#) on page 509.

3. Add settings for table structure; see §32.7.7 [Providing table structure information for DITA topic types](#) on page 916.

4. In your project configuration file, list the name of your new topic type:

```
[DITAContentModels]
DITAnewtype = any text here (ignored)
```

See §32.7.5 [Listing DITA topic type configuration files](#) on page 915.

5. Place `DITAnewtype.ini` where **Mif2Go** can find it; see §32.7.6 [Locating DITA topic type configuration files](#) on page 916.

32.7.2 Overriding settings in a DITA content model

You can override features of a built-in or previously defined DITA content model *without* creating a specialized type, by providing a content-model configuration file that lists *only* the differences from the original model. You can use this method to modify maps as well as topic types; see §32.7.4 [Overriding declarations in a DITA map content model](#) on page 915.

To override settings in a DITA content model:

1. Create a new `DITAtopictype.ini` configuration file from scratch, named for the topic type you are overriding. *Do not* use `dtd2ini` to generate this file from a DTD.
2. In configuration file `DITAtopictype.ini`, specify the name of the topic type you are overriding:

```
[Topic]
; TopicDerivation = name of type from which it is derived,
; either one of the defined types (topic, concept, task,
; reference, glossary, or map) or another specialized type
; for which an .ini is available.
TopicDerivation = topictype
```

`TopicDerivation` can be any of the built-in topic types (`topic`, `concept`, `task`, `reference`, `glossary`, `map`, or `bookmark`), or any specialized type for which a content-model configuration file named `DITAtopictype.ini` is available (see §32.7 [Specializing or modifying DITA topic types](#) on page 913). *Do not* use `TopicDerivation` in content-model configuration files generated by `dtd2ini`; those content models are always complete.

3. Other than a value for `TopicDerivation`, include settings in `DITAtopictype.ini` *only* for elements you are adding or modifying.
4. In your project configuration file, list the name of the topic type you are overriding:

```
[DITAContentModels]
topictype = any text here (ignored)
```

See §32.7.5 [Listing DITA topic type configuration files](#) on page 915.

5. Place `DITAtopictype.ini` where **Mif2Go** can find it; see §32.7.6 [Locating DITA topic type configuration files](#) on page 916.

For example, to change the `PUBLIC` declaration for `glossary` topics (to conform to XML requirements) without changing the declaration for any other topic type:

```
[Topic]
ModelName = glossary
```

```

TopicDerivation = glossary
TopicRoot = glossentry
PrologDType = "-//OASIS//DTD DITA Composite//EN"
PrologDTD = "database.dtd"

```

In your project configuration file:

```

[DITAContentModels]
glossary = my modified model for XMetaL (a comment)

```

32.7.3 Eliminating elements from a DITA content model

If you want to be able to tell **Mif2Go** *not* to use certain elements when unstructured FrameMaker text is parsed and element parents are interpolated, you can adjust the content model to remove those elements from the element sets, or alter their priority by listing them last in each element set. Bear in mind that the same set can be used for many elements. We advise not adding or removing any items, because that can result in invalid DITA. However, removal is safer than addition.

32.7.4 Overriding declarations in a DITA map content model

You can override the `PUBLIC` and `SYSTEM` IDs for a specialized map or bookmap the same way as for other topic types; see §32.7.2 [Overriding settings in a DITA content model](#) on page 914. However, for the maps **Mif2Go** generates, these declarations are about all you can change; the rest is hardwired.

To override declarations in a DITA map content model, create a new empty `DITAmapping.ini` configuration file. In this new configuration file specify the `PUBLIC` and `SYSTEM` IDs for a your specialized map. For example:

```

[Topic]
ModelName = map
TopicDerivation = map
PrologDType = "-//MYCO//DTD DITA MYCO Map//EN"
PrologDTD = "myco-map.dtd"

```

Also include the following setting in your project configuration file:

```

[DITAContentModels]
map = my company's modified map model (a comment)

```

See §32.7.5 [Listing DITA topic type configuration files](#) on page 915.

32.7.5 Listing DITA topic type configuration files

When you provide a `DITAtopictype.ini` configuration file, you must list the name of the topic type in your project configuration file, so **Mif2Go** knows you are specializing, and knows to look for the name of the specialized configuration file.

To list specialized topic types, in your project configuration file specify the following:

```

[DITAContentModels]
DITAtopictype = any text here (ignored)

```

Give each new type any alphanumeric name, except the name of a built-in type; that is, you may not name a *new* type `topic`, `concept`, `task`, `reference`, `map`, or (for DITA version 1.1) `glossary`. List the name of a built-in topic type *only* if you are overriding a feature of that topic type.

You can put whatever you want to the right of the equals sign; **Mif2Go** reads only the topic type name to the left of the equals sign.

Provide a `DITAtopicType.ini` configuration file named for each topic type you list; see §32.2.2 [Generating a content model from a DTD](#) on page 906. or §32.7.2 [Overriding settings in a DITA content model](#) on page 914. **Mif2Go** loads each listed `DITAtopicType.ini` configuration file at start-up, after initializing internal values for the built-in base topic types.

You do not have to list a topic type if the type is explicitly requested through an assignment to `[DITAOptions]DefTopic` (see §15.9.2.2 [Specifying a default DITA topic type](#) on page 525), or in a **DITATopic** marker, in which case the corresponding `DITAtopicType.ini` configuration file loads on demand. If the topic type information replaces one or more of the built-in types, this is the best way to load it.

If you create a new topic type that is derived from another new type, you can optionally list only the last topic type in the chain to get the whole batch loaded. Listing all types in the chain is harmless, but unnecessary.

32.7.6 Locating DITA topic type configuration files

By default, **Mif2Go** expects to find `DITA*.ini` configuration files in the project directory. To specify a different location for `DITA*.ini` configuration files for your project, include the following setting in your project configuration file:

```
[DITAOptions]
; SpecIniDir = path to add to names of specialized .inis,
; default "."
SpecIniDir = D:/path/to/myproj/config/
```

You can specify either a relative path or an absolute path for `SpecIniDir`. A relative path is relative to the project directory.

32.7.7 Providing table structure information for DITA topic types

You must include a section for each table type in each DITA topic-type content model, to provide information about table structure that cannot be abstracted from a DTD.

If you generated the content model for a specialized topic type from a DTD, and you plan to rerun **dttd2ini**, include in configuration file `dttd2ini.ini` any sections and settings you add for tables, in addition to including those sections in the content-model configuration file for the topic type.

In this section:

§32.7.7.1 [Mapping Mif2Go table types to configuration sections](#) on page 916

§32.7.7.2 [Omitting a table type from a derived topic type](#) on page 917

§32.7.7.3 [Assigning properties to Mif2Go table types](#) on page 917

§32.7.7.4 [Deriving a new table type](#) on page 917

32.7.7.1 Mapping Mif2Go table types to configuration sections

To provide a configuration for a **Mif2Go** table type, in the `DITAtopicType.ini` configuration file for the applicable topic type, map the table type name to a configuration section you provide. For example:

```
[TopicTables]
; Table name = name of configuration section that describes it.
property = PropertyTable
simple = SimpleTable
complex = ComplexTable
```

“Table name” is the name of a **Mif2Go** table type; see §15.6 [Converting tables to DITA XML](#) on page 510. Define here all table types supported by this topic type (other than those defined in the topic type from which this type was derived).

Because **dtd2ini** does not generate these sections, you must either include them in `dtd2ini.ini` as `[AddedSections]`, or add them to the generated content-model configuration file after **dtd2ini** produces it.

You can define variants for all table types that are supported by the topic type to which the `DITAtopictype.ini` configuration file applies. Multiple named **Mif2Go** table types can be mapped to variants of the same DITA table type.

32.7.7.2 Omitting a table type from a derived topic type

To undefine a **Mif2Go** table type in a derived topic type, set the table type name to `No`. For example:

```
[TopicTables]
complex = No
```

32.7.7.3 Assigning properties to Mif2Go table types

Give each DITA topic-type configuration section that contains table-type definitions the same name you assign to the table type in `[*Tables]`; see §32.7.7.1 [Mapping Mif2Go table types to configuration sections](#) on page 916. The examples in §F [Content model configuration](#) on page 1043 show all available settings for table types.

32.7.7.4 Deriving a new table type

Although you can assign the `strip` table type to omit table coding from DITA output for a particular FrameMaker table format (see §15.6.6 [Converting tables used only as image containers](#) on page 514), and build your own structure around the content of a table, in some cases it is better to derive a new table type with exactly the properties you need.

For example, you can add a derived `reference` topic type that includes a `properties` table type that has only two columns (`propvalue` and `propdesc`), replacing the original `reference` content model. Then you can list the new table type in project configuration section `[DITATables]`.

Suppose your FrameMaker table format is named *Commands*, and you want to define a new `properties` table type named `propval` for this format. You would create content-model configuration file `DITAprpval.ini` for the definition of `propval`.

In your project configuration file:

```
[DITAOptions]
SpecIniDir = path/to/content/models

[DITATables]
Commands = propval

[DITAContentModels]
propval = added propval table type
```

In `DITAprpval.ini`:

```
; Content model to modify reference.dtd by adding propval table type
[Topic]
TopicRoot = reference
TopicStart = title
TopicBody = refbody
PrologDType = "-//OASIS//DTD DITA 1.1 Reference//EN"
; Note -- the following setting must be all on one line:
```

```

PrologDTD =
    "http://docs.oasis-open.org/dita/v1.1/CS01/dtd/reference.dtd"
TopicDerivation = reference
ModelName = reference

[TopicTables]
propval = PropvalTable

[PropvalTable]
TableType = properties
ColCountMax = 2
HeadRowMax = 1
HeadRow = prophead
HeadCell1 = propvaluehd
HeadCell2 = propdeschd
Row = property
Cell1 = propvalue
Cell2 = propdesc

```

You can add as many variants of the property table type as you please, such as another property table with no header rows. Give each variant a new table type name (such as `propval`) but the same regular DITA `TableType`.

32.8 Extracting content-model debug information

You can have **Mif2Go** save tag-set information from a content model, for debugging purposes. The default is not to dump tag-set information. If the tag set is used more than once in processing a `FrameMaker` file, it is dumped only the first time.

To see what the tag set looks like for a content model:

```

[Topic]
; DumpToFile = name with optional path of file in which to dump the
; tagset information (including error lists) after loading.
DumpToFile = anyname.txt

```

You can specify any file name for `DumpToFile`, and optionally include a path. If you do not include a path, **Mif2Go** places the dump file in the project directory.

33 Overriding configuration settings

You can provide different configuration settings for individual FrameMaker files in a book, and you can also override configuration settings for one or more paragraphs, character spans, tables, graphics, or cross references within a FrameMaker file. Topics include:

§33.1 [Using a different configuration for selected files](#) on page 919

§33.2 [Overriding settings with markers or macros](#) on page 920

§33.3 [Overriding configuration settings with text](#) on page 931

See also:

§22.6.2 [Changing CSS files in the middle of a document](#) on page 689

33.1 Using a different configuration for selected files

If you need different configuration settings for one or more files in a book, you can create individual, file-specific configuration files.

In this section:

§33.1.1 [Providing configuration files for individual chapters](#) on page 919

§33.1.2 [Understanding precedence of configuration settings](#) on page 919

§33.1.3 [Updating a single chapter of a FrameMaker book](#) on page 920

33.1.1 Providing configuration files for individual chapters

To provide individual configuration files:

- Name each configuration file the same as the chapter file name, with extension `.ini`.
- Place individual configuration files in the same directory as the main configuration file for the project.
- Include in these chapter-specific configuration files *only* those settings that are different from settings in the main project configuration file.

When you run **Mif2Go** from the book file, the individual configuration files work in concert with the main configuration file; settings in an individual configuration file override the corresponding settings in the main configuration file, for that chapter file.

See also:

§30.4 [Including chapter-specific configuration files](#) on page 855

33.1.2 Understanding precedence of configuration settings

At run time **Mif2Go** builds a configuration for each FrameMaker file in your project, beginning with the most specific settings: those in any chapter-specific configuration file, if there is one. Next come settings in the project configuration file.

*Chain of
configuration
templates*

Next, if the chapter-specific configuration file includes a value for `[Templates]Configs` (see §30.2 [Referencing configuration files and templates](#) on page 851), settings in the referenced configuration template (and any additional templates chained to it) are applied. If the chapter-specific configuration file does not reference a configuration template, next come settings in any configuration template referenced by the project configuration file; then on up the chain from that template. [Table 33-1](#) shows the precedence of settings in configuration files and templates.

Table 33-1 Precedence of settings in configuration files and templates

Precedence	Configuration file	Description
Highest	<i>chapter.ini</i>	For a book, configuration file (if any) for a single FrameMaker file named <i>chapter.fm</i>
	<i>_m2*.ini</i>	Project configuration file
	<i>chaptemplate.ini</i> or <i>doctemplate.ini</i>	Template referenced by <i>chapter.ini</i> , if any
		Template referenced by <i>_m2*.ini</i> via [Templates]Document if no such template is referenced by <i>chapter.ini</i> (or no <i>chapter.ini</i> is present)
	<i>projtemplate.ini</i>	Template referenced by <i>_m2*.ini</i> if no template is referenced by <i>chapter.ini</i> (or no <i>chapter.ini</i> is present)
	<i>commontemplate1.ini</i>	Template referenced by <i>chaptemplate.ini</i> or by <i>projtemplate.ini</i> , whichever is used
	<i>commontemplateN.ini</i>	Template referenced by <i>commontemplateN-1.ini</i>
Lowest	Default value	Whatever the Mif2Go default value is for the setting in question

A chain of configuration templates, if any, is applied to the source either from *chapter.ini* (preferentially) or from the project configuration file, but not from both. In either case, settings from the templates are applied after settings from the project configuration file, which are applied after settings from the chapter configuration file. For the same setting with different values in different configuration files or templates, the value in the most specific file takes precedence. See §30.6.3 [Chaining configuration templates](#) on page 863.

33.1.3 Updating a single chapter of a FrameMaker book

To update a single chapter for which you have created an individual configuration file:

1. Make sure you have already set up the project from the book file; see §3.3 [Creating a Mif2Go conversion project](#) on page 78.
2. Keeping the book file open, run **Mif2Go** from the chapter file.

Mif2Go determines that this is a chapter of a book, and builds both chapter-related files *and* book-related files. For HTML output, if a change you make in a chapter results in a change of file name for a split file, and another chapter has a cross reference or hypertext link to a marker in that split file, **Mif2Go** updates the link in files produced from the other chapter.

33.2 Overriding settings with markers or macros

To change the value of a configuration setting partway through a FrameMaker file, you assign a new value to a configuration variable. You can insert a FrameMaker marker that contains the assignment or (for some settings) define a **Mif2Go** macro that includes the assignment. Both methods allow you to shift configuration values back and forth within the same FrameMaker file.

In this section:

§33.2.1 [Determining the extent of a configuration override](#) on page 921

§33.2.2 [Overriding settings with configuration markers](#) on page 921

- §33.2.3 [Overriding settings with macros](#) on page 921
- §33.2.4 [Assigning values to configuration variables](#) on page 922
- §33.2.5 [Adding a new configuration setting on the fly](#) on page 923
- §33.2.6 [Assigning a macro or variable to a configuration variable](#) on page 923
- §33.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 923
- §33.2.8 [Overriding fixed-key configuration settings](#) on page 924
- §33.2.9 [Overriding variable-key configuration settings](#) on page 925
- §33.2.10 [Assigning HTML table and graphic groups with overrides](#) on page 930

33.2.1 Determining the extent of a configuration override

An override to a configuration setting can affect either a single item in your document (a temporary override), or a series of items (a persistent override), depending on the syntax you use for the override; see §33.2.4 [Assigning values to configuration variables](#) on page 922. **Mif2Go** does not store either persistent or temporary overrides in your configuration file. The configuration file always retains the original values of the settings.

<i>Persistent overrides</i>	A <i>persistent override</i> stays in effect until changed by another override of the same setting, or until the end of the FrameMaker file in which the override occurs, whichever comes first. To apply a persistent override, insert a marker in your document just before the place where you want the override to take effect; and (optionally) later, another marker to reverse the effect. For certain fixed-key settings, you can include a configuration override in a regular Mif2Go macro instead of in a marker; see §33.2.3 Overriding settings with macros on page 921.
<i>Temporary overrides</i>	A <i>temporary override</i> affects only one instance of the item (text, table, or graphic) to which the setting applies. To apply a temporary override, you insert a marker in or just before the item to which the override should apply. Temporary overrides can be applied only to variable-key settings; see §33.2.7 Understanding fixed-key vs. variable-key settings on page 923.

33.2.2 Overriding settings with configuration markers

To change a configuration setting mid-document with a configuration marker, you must first define one (or more) of the following custom marker types in FrameMaker:

Config	applies either to HTML or to RTF, wherever the setting is applicable
HTMConfig	applies only to HTML output; ignored for RTF output
RTFConfig	applies only to RTF output; ignored for HTML output.

To add these marker types to your document, use the FrameMaker *Edit Custom Marker Type* dialog, reached via **Special > Marker > Marker Type: Edit...**; see §29.2 [Adding custom marker types](#) on page 832.

To change the value of a configuration setting partway through your document, insert a configuration marker (**Config**, **HTMConfig**, or **RTFConfig**) at the place where you want the value to change, and supply a configuration-variable assignment as content for the marker, according to the syntax and usage described in §33.2.4 [Assigning values to configuration variables](#) on page 922.

33.2.3 Overriding settings with macros

To change a configuration setting mid-document with a macro, you must include a configuration-variable assignment either in a code-type FrameMaker marker or (for

persistent overrides only) in a configuration macro included in your configuration file or macro library.

The macro override choices apply as follows:

HTML Code marker	HTML output only; ignored for RTF output
Code marker	HTML or RTF output, wherever the setting is applicable
Mif2Go macro	HTML or RTF output, wherever the setting is applicable, but only for persistent overrides.

To change the value of a configuration setting with a macro in a marker, insert a marker of type **Code** or **HTML Code** at the place where you want the value to change, and supply as content for the marker a configuration-variable assignment constructed as described in §33.2.4 [Assigning values to configuration variables](#) on page 922.

For persistent overrides only, you can include the configuration-variable assignment in a configuration macro that applies the directive based on some condition; see §33.2.1 [Determining the extent of a configuration override](#) on page 921.

33.2.4 Assigning values to configuration variables

A configuration-variable assignment can be any of the following, depending on the context and the extent of the configuration override:

<u>Context</u>	<u>Persistent override</u>	<u>Temporary override</u>
*Config marker	[<i>Section</i>]Key=Value	[<i>Section</i>]=Value
Mif2Go macro	\$\$[<i>Section</i>]Key=Value	\$\$[<i>Section</i>]=Value

where the components of the assignment are as follows: :

<i>Section</i>	Name of the configuration-file section where the setting belongs
<i>Key</i>	Keyword whose value you want to change, or the format or object whose properties you want to change; omit for temporary overrides
<i>Value</i>	New value for the setting.

When you assign a value to a configuration variable, observe the following:

- Spaces around [*Section*] are optional.
- *Section* is not case sensitive.
- In a macro, [*Section*] must be prefixed with \$\$; in a ***Config** marker, the prefix is optional.
- Include *Key* only for a persistent override; see §33.2.1 [Determining the extent of a configuration override](#) on page 921.
- *Key* is case sensitive for variable-key settings.
- *Key* may not use wildcards.
- *Key* must be enclosed in quotes if *Key* contains any spaces or non-alphanumeric characters.
- If *Key* requires an on/off value, **Mif2Go** recognizes “1” (numeral one), “Yes”, and “True” as on, and “0” (zero), “No”, and “False” as off.
- In a macro, if *Value* is a text string, *Value* must be enclosed in quotes. In a ***Config** marker, quotes around text values are optional; if present, **Mif2Go** removes them. Therefore, if the value to be assigned actually contains quotes at both ends, you must double them for assignment in a ***Config** marker. For example:

HTMConfig: [StyleTextStore]=" "a quoted phrase" "

- If *value* includes the name of a macro or macro variable, whether that name should be enclosed in quotes depends on the context; see §33.2.6 [Assigning a macro or variable to a configuration variable](#) on page 923.

33.2.5 Adding a new configuration setting on the fly

Besides overriding existing settings in the configuration file, you can use a configuration-variable assignment to specify a persistent override for a setting that is not even present in your configuration file, provided both of the following are true:

- *[Section]* is listed as subject to overrides in one of [Table 33-2](#) through [Table 33-6](#).
- *Key* is a valid key for the section.

If the section is not listed, or the key is not valid for the section, the setting you specify is treated instead as an error, with value “0” (zero).

33.2.6 Assigning a macro or variable to a configuration variable

When you assign a value to a configuration variable, and the value includes the name of a macro or a macro variable, whether or not that name should be enclosed in quotes depends on the context:

- In a ***Config** marker, a value is always assigned literally, *as is*, so you can either include or omit quotes around the name of a macro or variable.
- In a macro, a value is assigned literally *only* if it is enclosed in quotes. If the value includes a macro name, the entire value should be quoted. Such a value may not contain a quote.

For example:

HTMConfig: [ParaStyleCodeAfter]=<\$macafter>

HTML Macro: <\$\$[ParaStyleCodeAfter]="<\$macafter">">

*Angle brackets
get processed in
a macro*

When you assign a value to a configuration variable in a macro, and the value contains any < or > characters (angle brackets), absent enclosing quotes **Mif2Go** processes each angle bracket as the start or end of a macro, instead of assigning the entire value as a string. That is, **Mif2Go** would try to figure out if maybe the string is something else first. When the value includes a > character that it is not in quotes, the macro ends prematurely. In this example:

<\$\$[ParaStyleCodeAfter]=<hr>>

Mif2Go would assign only <hr to the configuration variable, because the > after <hr would be taken as the end of the macro; and then **Mif2Go** would drop the real ending > into the current text.

*Unquoted
variables are
evaluated in a
macro*

When you assign a macro variable to a configuration variable in a macro:

- Enclose the macro variable name in quotes if you want the macro variable to be evaluated later, at run time.
- Do not enclose the macro variable name in quotes if you want the macro variable to be evaluated immediately, so the configuration setting gets the current value of the macro variable instead of just its name.

33.2.7 Understanding fixed-key vs. variable-key settings

The settings in some **Mif2Go** configuration sections are global in scope, and use *fixed keys*: predefined keywords to which you can assign values. The settings in other configuration sections use *variable keys*: the names of formats, tables, or graphics in your

document. You can override some settings in most fixed-key sections, and all settings in most variable-key sections.

*Fixed-key
configuration
sections*

Configuration sections such as [HTMLOptions] have a set of predefined keywords, and the value you assign to a given keyword usually applies to the entire document. You can change the value of a fixed-key setting only with a persistent override, where you name the key whose value is to be overridden. Temporary overrides do not apply to fixed-key settings; see §33.2.1 [Determining the extent of a configuration override](#) on page 921. [Table 33-2](#) on page 925 lists the fixed-key configuration sections that include settings subject to override.

*Variable-key
configuration
sections*

Configuration sections such as [HelpStyles] or [GraphAlign] use format names or object identifiers as keys, where the key name is one of the following:

- a character, paragraph, or cross-reference format name; the value applies only to text in the named format
- a graphic ID, table ID, or table format name, or a named group of graphics or tables; the value applies only to the named table, graphic, or group.

You can use either persistent overrides or temporary overrides for most variable-key settings. You can override settings in the variable-key sections listed in the following tables:

[Table 33-3 Text configuration sections subject to overrides](#) on page 926

[Table 33-4 Cross-reference sections subject to overrides](#) on page 928

[Table 33-5 HTML table sections subject to overrides](#) on page 928

[Table 33-6 HTML graphic sections subject to overrides](#) on page 930

33.2.8 Overriding fixed-key configuration settings

An override to a fixed-key configuration setting stays in effect until the end of the current FrameMaker file, or until changed again by another configuration marker or configuration-variable assignment to the same setting. You can override some (but not all) settings in the configuration sections listed in [Table 33-2](#) on page 925. For example, to switch mid-file to turning on revision tracking in Word:

<u>Configuration setting</u>	<u>RTFConfig override</u>
[WordOptions] RevTrack = No	[WordOptions]RevTrack=Yes

Only persistent overrides work for fixed-key settings; temporary overrides do not work. Also, persistent overrides work only for fixed-key settings that do not have to apply to an entire FrameMaker file. For instance, it would make no sense to try to change, in the middle of a file, the value of [Setup]ApplyTemplateFile; applying a conversion template is a one-time function that takes place before Mif2Go processes the file content. Other settings such as [WordOptions]SideHeads affect margins, and must apply to an entire file.

For example, suppose your document contains white text that you do not want to display in HTML output; so, in the configuration file you include the following setting:

```
[HTMLOptions]
HideWhiteText=Yes
```

But suppose your document also includes some white headings, in paragraph format *Head1*, that are displayed inside a black box provided in a Frame Below with negative space. You could use macros to override the HideWhiteText setting for these headings:

```
[ParaStyleCodeBefore]
Head1=<$$[HTMLOptions]HideWhiteText=0>
```



```
[ParaStyleCodeAfter]
Head1=<$$[HTMLOptions]HideWhiteText=1>
```

If you are producing HTML output, the only way to specify attributes for <body> is with a configuration override. For example, placing a marker at the beginning of the second topic in a FrameMaker file:

```
[Attributes]body= onload='prettyPrint()'
```

Then at the beginning of the fourth topic:

```
[Attributes]body=
```

The effect would appear in HTML output for the second and third topics, but not the fourth. The marker affects the output file for the topic in which it is included, and continues until set otherwise.

You can also override a fixed-key setting with a configuration-variable assignment in a regular **Mif2Go** macro instead of in a marker. See §28.9.3 [Surrounding or replacing text with code or macros](#) on page 822.

Table 33-2 Fixed-key configuration sections subject to overrides

Fixed-key configuration section *	HTML/XML	Word	WinHelp
[Attributes]	Yes		
[Base]	Yes		
[CharClasses]	Yes		
[Defaults]		Yes	Yes
[Setup]	Yes	Yes	Yes
[GraphExport]	Yes	Yes	Yes
[Graphics]	Yes	Yes	Yes
[HelpBrowse]			Yes
[HelpContents]			Yes
[HelpOptions]			Yes
[HTMLOptions]	Yes		
[Inserts]	Yes	Yes	Yes
[JavaHelpOptions]	Yes		
[LocalTOC]	Yes		
[Macros]	Yes		
[MSHtmlHelpOptions]	Yes		
[NavigationMacros]	Yes		
[OmniHelpOptions]	Yes		
[Options]	Yes	Yes	Yes
[ParaClasses]	Yes		
[Tables]	Yes		
[Trails]	Yes		
[WordOptions]		Yes	

* Some settings cannot be overridden in these sections; you might have to experiment.

33.2.9 Overriding variable-key configuration settings

In this section:

§33.2.9.1 [Overriding paragraph and character format properties](#) on page 926

- §33.2.9.2 [Overriding cross-reference properties](#) on page 928
- §33.2.9.3 [Overriding table properties for HTML](#) on page 928
- §33.2.9.4 [Overriding graphic properties for HTML](#) on page 929

33.2.9.1 Overriding paragraph and character format properties

You can override character and paragraph format settings in the configuration sections listed in [Table 33-3](#). For example, to specify new properties for a single paragraph in HTML, you could insert in the paragraph an **HTMConfig** marker with content different from the default:

```
[HTMLParaStyles]Size5 Bold
```

In a macro, you would specify:

```
<$$[HTMLParaStyles]Size5 Bold>
```

Temporary overrides

Most configuration settings for text properties can apply to either a paragraph format or a character format. Temporary overrides lack a key to name the format to be affected; therefore, for a temporary override, where in the text you place the configuration marker with respect to paragraph and character formats is critical:

- A temporary-override marker placed in a paragraph but not in a character span affects the entire paragraph, even if character spans intervene or end the paragraph.
- If a paragraph starts with a character span, whether you place the marker for a temporary override before or after the start of the character format determines whether the override applies to the paragraph or to the character span. Check the FrameMaker status bar when you insert the marker, to see which format is shown.
- A temporary-override marker placed in the span of a character format affects only the current span; the end of the character span resets the override, and it has no further effect, even if the same character format is used again in the same paragraph.

Persistent overrides

A persistent override affects the next instance of the paragraph or character format named by the *Key* in `[Section]Key=Value`, or the current instance if the marker is placed in a matching paragraph or character span; plus all subsequent instances in the same FrameMaker file, unless changed again by a later override.

Markers in replaced text are ignored

For `[ParaStyleCodeReplace]`, if placement code is already in effect because it was specified in the configuration file, any configuration marker in the replaced text is ignored. This means you cannot use a temporary override in a configuration marker for the replacement; instead you must use a persistent override that names the format to be replaced, and insert the configuration marker before the text to be replaced.

Place overrides to code with care

For `[HTMLParaStyles]` and `[HTMLCharStyles]`, temporary overrides to `Delete` assignments must be inserted before the first text in the affected paragraph or character span. Persistent overrides should be placed before the affected paragraph or character span.

Table 33-3 Text configuration sections subject to overrides

Text configuration section	HTML/XML	Word	WinHelp
[AnumCodeAfter]	Yes	Yes	Yes
[AnumCodeBefore]	Yes	Yes	Yes
[CharStyleCodeAfter]	Yes	Yes	Yes
[CharStyleCodeBefore]	Yes	Yes	Yes
[CharStyleCodeEnd]	Yes	Yes	Yes
[CharStyleCodeReplace]	Yes	Yes	Yes
[CharStyleCodeStart]	Yes	Yes	Yes

Table 33-3 Text configuration sections subject to overrides (continued)

Text configuration section	HTML/XML	Word	WinHelp
[CharStyleCSS]	Yes		
[CharTags]	Yes		
[ExtrBottom]	Yes		
[ExtrHead]	Yes		
[ExtrReplace]	Yes		
[ExtrTitle]	Yes		
[ExtrTop]	Yes		
[HelpBrowsePrefixStyles]			Yes
[HelpCntStyles]			Yes
[HelpContentsLevels]	Yes		
[HelpJumpFileStyles]			Yes
[HelpKeywordStyles]			Yes
[HelpMacroStyles]			Yes
[HelpRefStyles]			Yes
[HelpReplacements]			Yes
[HelpStyles]			Yes
[HelpSuffixStyles]			Yes
[HelpTitleSufStyles]			Yes
[HelpTopicBuildStyles]			Yes
[HelpWindowStyles]			Yes
[HTMLCharStyles]	Yes		
[HTMLParaStyles]	Yes		
[LocalTOCLevels]	Yes		
[ParaStyleCodeAfter]	Yes	Yes	Yes
[ParaStyleCodeBefore]	Yes	Yes	Yes
[ParaStyleCodeEnd]	Yes	Yes	Yes
[ParaStyleCodeReplace]	Yes	Yes	Yes
[ParaStyleCodeStart]	Yes	Yes	Yes
[ParaStyleCSS]	Yes		
[ParaTags]	Yes		
[SecWindows]	Yes		
[StyleCellAbbr]	Yes		
[StyleCellAttribute]	Yes		
[StyleCellAxis]	Yes		
[StyleCellScope]	Yes		
[StyleCodeStore]	Yes	Yes	Yes
[StyleFilePrefix]	Yes		
[StyleFileSuffix]	Yes		
[StyleLinkSrc]	Yes		
[StyleMetaName]	Yes		
[StyleParaLinkClass]	Yes		
[StyleRowAttribute]	Yes		
[StyleTextStore]	Yes		

Table 33-3 Text configuration sections subject to overrides (continued)

Text configuration section	HTML/XML	Word	WinHelp
[StyleTitlePrefix]	Yes		
[StyleTitleSuffix]	Yes		
[StyleTrailPrefix]	Yes		
[StyleTrailSuffix]	Yes		
[Targets]	Yes		
[TrailLevels]	Yes		
[WordCntStyles]		Yes	
[WordReplacements]		Yes	
[WordStyles]		Yes	

33.2.9.2 Overriding cross-reference properties

You can use configuration markers and configuration-variable assignments to override settings in [XrefStyles] and in the HTML [XrefStyleLinkSrc] section; see [Table 33-4](#).

A temporary override to a cross-reference format affects the next cross reference after the configuration marker.

A persistent override affects the next cross reference in the format named by the *Key* in [Section]Key=Value, and all subsequent instances in the same FrameMaker file, unless changed again by a later override.

Table 33-4 Cross-reference sections subject to overrides

Cross-reference section	HTML/XML	Word	WinHelp
[XrefStyleLinkSrc]	Yes		
[XrefStyles]	Yes	Yes	Yes

33.2.9.3 Overriding table properties for HTML

You can use configuration markers and configuration-variable assignments to override settings in the HTML [Table*] sections listed in [Table 33-5](#).

A temporary override to a table affects the entire table within which a configuration marker is placed; or the next table, if the marker is not in a table.

A persistent override affects the next table with the ID or table format, or in the table group, named by the *Key* in [Section]Key=Value; or the current instance, if the marker is placed in a matching table. For table formats and groups, the override also affects all subsequent matching instances in the same FrameMaker file, unless changed again by a later override.

Table 33-5 HTML table sections subject to overrides

Table configuration section
[TableAccess]
[TableAfterMacros]
[TableAttributes]
[TableBeforeMacros]
[TableBodyAttributes]

Table 33-5 HTML table sections subject to overrides (continued)**Table configuration section**

```
[TableCellAttributes]
[TableCellEndMacros]
[TableCellStartMacros]
[TableEndMacros]
[TableFooterAttributes]
[TableGroup]
[TableHeaderAttributes]
[TableIndents]
[TableReplaceMacros]
[TableRowAttributes]
[TableRowEndMacros]
[TableRowStartMacros]
[TableSizing]
[TableStartMacros]
[TableUseRowColor]
```

See also:

§33.2.10 [Assigning HTML table and graphic groups with overrides](#) on page 930

33.2.9.4 Overriding graphic properties for HTML

You can use configuration markers and configuration-variable assignments to override any variable-key settings and some fixed-key settings in the HTML [Graph*] sections listed in [Table 33-6](#).

A temporary override to a graphic affects the next graphic after the point in your document where you insert the marker.

Note: Configuration markers placed in text frames within graphic frames do not work.

A persistent override affects the next graphic with the ID or in the graphic group named by the *Key* in [Section]Key=Value; and for graphic groups, all subsequent matching instances in the same FrameMaker file, unless changed again by a later override.

One additional section, [GraphGroup], is handled differently from the rest. For [GraphGroup], the directive assigns the graphic to a named graphic group. You can use only a temporary setting applied with a ***Config** marker, not a macro, to specify the graphic group.

Overriding the overrides

To override path settings both in [GraphFiles] and in configuration markers with whatever you specify for [Graphics]GraphPath:

```
[Graphics]
; GraphPathOverrides = No (default) or Yes (overrides any path
; in Config markers and in [GraphFiles], adding GraphPath
; and using FixGraphSpaces)
GraphPathOverrides=Yes
```

When GraphPathOverrides=Yes, **Mif2Go** uses the path to graphics specified by GraphPath (see §31.3.1.1 [Specifying graphics location for HTML](#) on page 887) instead of any path (or lack of a path) specified in [GraphFiles] (see §31.3.1.2 [Substituting graphics files for HTML](#) on page 888) or in a ***Config** marker with content:

```
[GraphFiles]=filename
```

Also, **Mif2Go** replaces with underscores any spaces in file names of referenced graphics; see §31.3.1.4 [Using original files and image sizes for referenced graphics](#) on page 889.

Table 33-6 HTML graphic sections subject to overrides

Graphic configuration section

```
[GraphAlign]
[GraphALT]
[GraphAttr]
[GraphDpi]
[GraphEndMacros]
[GraphFiles]
[GraphGroup]
[GraphHigh]
[GraphIndents]
[GraphParaAlign]
[GraphReplaceMacros]
[GraphRightSpacers]
[GraphScale]
[GraphStartMacros]
[GraphWide]
```

See also:

§23.7 [Specifying HTML image attributes](#) on page 718

§31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896

§33.2.10 [Assigning HTML table and graphic groups with overrides](#) on page 930

33.2.10 Assigning HTML table and graphic groups with overrides

Two variable-key configuration sections, [TableGroup] and [GraphGroup], are handled differently from the rest. For [TableGroup] and [GraphGroup], a configuration marker assigns the table or graphic to a named group. You can use only a temporary override applied with a ***Config** marker, not a macro, to specify the group name. Any key included in the marker is ignored.

Table groups If you put a [TableGroup] configuration marker in each table that should be assigned to a given group, you can specify settings for all members of that table group in a [Table*] section in the configuration file, without having to look up any table IDs.

Note: Each table can belong to only one table group.

See also:

§24.2.2 [Creating table groups](#) on page 729

§33.2.9.3 [Overriding table properties for HTML](#) on page 928

Graphic groups If you put a [GraphGroup] configuration marker just before each graphic that should be assigned to a given group, you can specify settings for all members of that graphic group in a [Graph*] section in the configuration file, without having to look up graphic IDs.

Note: Each graphic can belong to only one graphic group.

See also:

§23.5.1.2 [Using markers to assign properties to graphics](#) on page 709

§33.2.9.4 [Overriding graphic properties for HTML](#) on page 929

33.3 Overriding configuration settings with text

To override configuration settings on the fly, you can include a configuration setting in your document as text, give it a unique paragraph format, and assign that format a special property. This method is an alternative to inserting **Config** or **HTMConfig** or **RTFConfig** markers in your document, and it works the same way. See §33.2.2 [Overriding settings with configuration markers](#) on page 921.

To make a FrameMaker paragraph act as a configuration override:

```
[HTMLParaStyles] or [WordStyles] or [HelpStyles]
; Config (and HTMConfig or RTFConfig) use the contents of the para as
; though it is a set of Config markers, each ending with a hard
; return, but also allow the normal .ini syntax with [Sections] on
; their own lines, and comments.
ParaFmt = Config Delete
```

Property **HTMConfig** is effective only in HTML output types, property **RTFConfig** is effective only in RTF output types, and where applicable, property **Config** is effective in both.

When you also assign property **Delete**, **Mif2Go** removes the paragraph from the actual text stream, so the text does not appear in the output.

The content of each paragraph in a format assigned the **Config** (or **HTMConfig** or **RTFConfig**) property is treated as a configuration override, or a series of configuration overrides, provided the content:

- conforms to configuration syntax
- specifies settings that are subject to overrides.

See §33.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 923.

You have two choices of syntax for ***Config** paragraph content; you can intermix them in the same paragraph:

- | | |
|-----------------------|---|
| <i>File syntax:</i> | Make the paragraph look like a configuration-file section, with a hard return at the end of each line (although a hard return is not required after the last line). You can include multiple configuration sections, and also include comment lines that start with a semicolon; see §4.4 Understanding the rules for configuration settings on page 102. |
| <i>Marker syntax:</i> | Use the same syntax as for *Config markers; see §33.2.4 Assigning values to configuration variables on page 922. Place a hard return at the end of each override. |

For example, a ***Config** paragraph that precedes an anchored frame that contains a graphic might provide the name of a different graphic to substitute for the one in your document:

```
[GraphFiles]
=Screen1.gif
```

The content of the paragraph could just as well look like this:

```
[GraphFiles]=Screen1.gif
```

The result works exactly like the same content put in markers at the same location in your document, and is subject to the same limitations as for markers, except there is no length limit to the content.

If your FrameMaker document is subject to frequent updates that might result in accidentally deleting markers, using a `*Config` paragraph instead makes the overrides less likely to be lost. For example, if you use conditional text to hide `*Config` paragraphs while you are editing the document, FrameMaker warns you before deleting the hidden text.

Note: For graphics in anchored frames, FrameMaker 7.0 and later versions provide yet another way to override settings; see §31.4.2 [Overriding graphics settings with FrameMaker object attributes](#) on page 896.

34 Automating Mif2Go conversions

Mif2Go supports several techniques for single-sourcing FrameMaker documents, and for automating conversion workflow. This section includes the following topics:

- §34.1 [Preparing documents for single-sourcing](#) on page 933
- §34.2 [Converting a single chapter of a book](#) on page 937
- §34.3 [Considering ways to automate conversions](#) on page 937
- §34.4 [Executing operating-system commands](#) on page 937
- §34.5 [Supplying run-time values for user variables](#) on page 941
- §34.6 [Supporting document review in Word](#) on page 943
- §34.7 [Converting autonumbers for database systems](#) on page 944
- §34.8 [Renaming output files for automated systems](#) on page 946

See also:

- §35 [Producing deliverable results](#) on page 955
- §36 [Converting via runfm](#) on page 979
- §37 [Converting via DCL](#) on page 995
- §38 [Generating intermediate output](#) on page 1005

34.1 Preparing documents for single-sourcing

You can fine-tune a FrameMaker document for single-sourcing, so **Mif2Go** can produce both print and on-line versions with no intermediate text editing or format tweaking. This section describes several techniques:

- §34.1.1 [Using character formats to identify Help elements](#) on page 933
- §34.1.2 [Using markers to add links and instructions](#) on page 935
- §34.1.3 [Using conditional text to differentiate output](#) on page 936
- §34.1.4 [Importing formats and conditional text settings](#) on page 936

34.1.1 Using character formats to identify Help elements

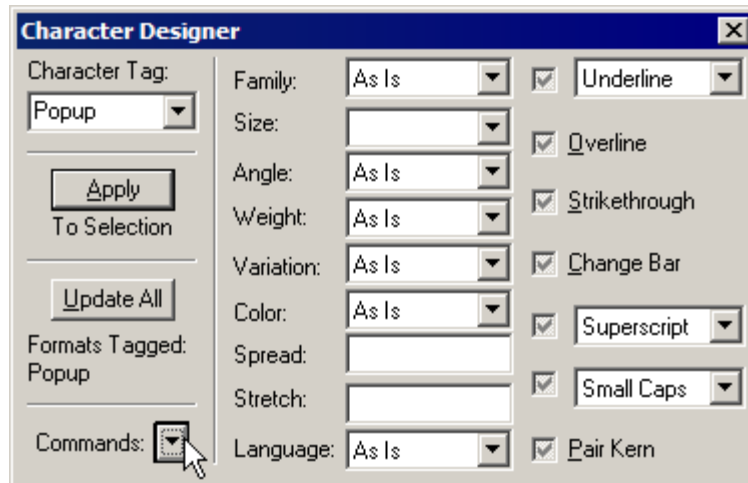
You can use character-format tagging to identify text you want to trigger help-specific actions. The character format need not have any effect on the text in FrameMaker; the format *name* is what **Mif2Go** goes by.

For example, to identify pop-up hotspots, you might create a FrameMaker character format called *Popup*, and apply it to each text item that should act as a pop-up hotspot in your help file. Suppose your document contains the following text:

As the fund-raising campaign progresses you will have an opportunity to visit Northcoast Redwoods and see for yourself the beauty of this forest.

Suppose you want “Northcoast Redwoods” to be a pop-up hotspot when you convert the document to Help:

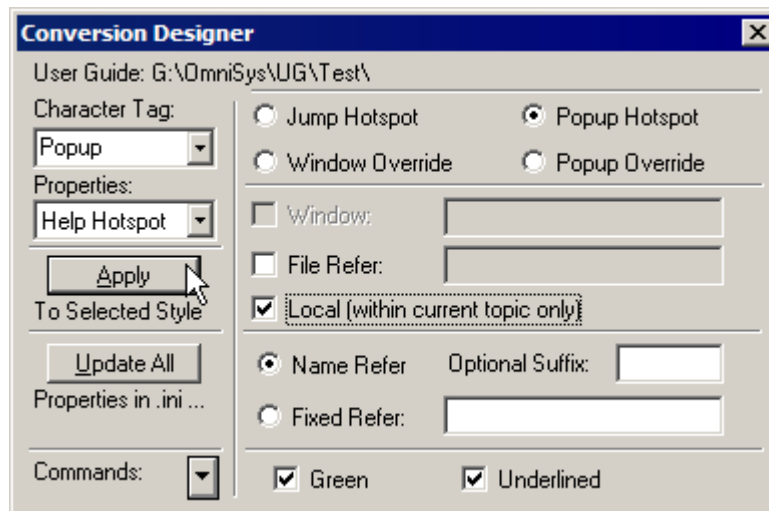
1. Use the FrameMaker Character Designer **Set Window to As Is** command to define character format *Popup* as shown in [Figure 34-1](#) on page 934.

Figure 34-1 Defining character format Popup

2. Apply character format *Popup*—which you have defined in Character Designer to be entirely “As Is”—to the phrase “Northcoast Redwoods”. The appearance of the text does not change in your FrameMaker document.
3. Add this setting to your project configuration file:

```
[HelpStyles]
Popup=PopOver Green
```

Or, create the same setting in Conversion Designer, as shown in [Figure 34-2](#).

Figure 34-2 Defining a character format for pop-up hotspots

When **Mif2Go** converts your document, the text in the resulting Help file looks something like [Figure 34-3](#).

Figure 34-3 Pop-up hotspot in WinHelp

As the fund-raising campaign progresses you will have an opportunity to visit [Northcoast Redwoods](#) and see for yourself the beauty of this forest.

Clicking [Northcoast Redwoods](#) pops up whatever text or graphics you define in your Help file to be a target for this pop-up.

See also:

§8.9.9 [Using the same content for both normal topics and pop-ups](#) on page 278

§8.9.11.5 [Assigning properties to alternative jumps and pop-ups](#) on page 281

34.1.2 Using markers to add links and instructions

You can use markers in a FrameMaker file to provide features required for assorted delivery formats:

[Hypertext alerts and navigation links](#)

[Reference strings in WinHelp](#)

[Code to execute in HTML](#)

[Conversion instructions](#)

*Hypertext alerts
and navigation
links*

To insert a hypertext marker:

1. Position the cursor in FrameMaker text where you want the marker.
2. Choose **Special > Hypertext...** from the FrameMaker top menu bar; the *Hypertext* dialog opens.
3. From the **Command:** list select the command for the type of hypertext marker you want to insert; typical hypertext markers and their corresponding FrameMaker hypertext commands are as follows:

<u>Marker type</u>	<u>FrameMaker hypertext command</u>
alert	Alert
alerttitle	Alert with Title
gotolink	Jump to Named Destination
message openfile	Message Client <i>(closest fit)</i>
message URL	Go to URL
newlink	Specify Named Destination
openlink	Open Document

The marker type name for the command appears in the **Syntax:** box, followed by a space. Above the **Syntax:** box you should see an example of the syntax to use with that marker type.

4. In the **Syntax:** box, after the space type a unique name for the link target, along with any other parameters required for the marker type.
5. Click **New Hypertext Marker** to insert the marker and dismiss the *Hypertext* dialog.

*Reference strings
in WinHelp*

To add reference strings as hypertext **newlinks** in WinHelp, see §8.9.5 [Using hypertext links for jumps and pop-ups](#) on page 276.

*Code to execute
in HTML*

To include JavaScript or **Mif2Go** macro code in HTML, see §29.7 [Inserting code or text with markers](#) on page 842.

*Conversion
instructions*

To embed conversion instructions in a FrameMaker document, see §33.2 [Overriding settings with markers or macros](#) on page 920.

34.1.3 Using conditional text to differentiate output

FrameMaker's conditional text feature is a powerful aid to single-sourcing. Conditional text allows you to create differences in content between printed documents and various on-line formats. You can do this with condition tags; for example:

PrintOnly	Content that should appear only in the printed document
HelpOnly	Content that should appear only in on-line Help

Content that should be present both in print and on line would be left **Unconditional**.

Although some people advise tagging no less than a paragraph, we have had no trouble applying condition tags to phrases, to individual words, and even to frame anchors, to include different graphics for different uses.

Mif2Go can import **Conditional Text Settings** from a FrameMaker template, to show only the conditions appropriate for a particular conversion; see §2.4 [Importing formats from a conversion template](#) on page 67.

See also:

- §3.4.1 [Importing formats from a FrameMaker template](#) on page 79
- §30.7 [Applying FrameMaker conversion templates](#) on page 863
- §34.1.4 [Importing formats and conditional text settings](#) on page 936

34.1.4 Importing formats and conditional text settings

You can import formats from a FrameMaker template to create alternate output.

To tell **Mif2Go** which template to use and what to import, do one of the following:

- [Specify imports at set-up time](#)
- [Specify imports in the configuration file](#)
- [Import manually for command-line conversion.](#)

*Specify imports at
set-up time*

When you first set up your **Mif2Go** conversion project in FrameMaker, use the *Set Up* dialog to specify the template and check the formats you wish to import; see §3.4.1 [Importing formats from a FrameMaker template](#) on page 79.

*Specify imports in
the configuration
file*

If you have already set up your **Mif2Go** conversion project, and you intend to run the conversion from FrameMaker, edit your project configuration file to specify template and formats; see §30.7 [Applying FrameMaker conversion templates](#) on page 863.

*Import manually
for command-line
conversion*

If you intend to run your **Mif2Go** conversion from the command line (see §37 [Converting via DCL](#) on page 995), you must provide your document in `.mif` format:

1. When you finish editing the print document, save it, but leave it open in FrameMaker. The `.fm` copy you just saved is the one you will use for revisions later.
2. Open the Help template or HTML template, shift to the open print document, and import formats from the template.
3. Regenerate the file, then save as `.mif`; this is the `.mif` copy you will use for **Mif2Go** to convert to on-line help or HTML. Close the file *without* saving again, so that your print document remains unchanged.

See also:

- §2.4 [Importing formats from a conversion template](#) on page 67
- §30.7 [Applying FrameMaker conversion templates](#) on page 863
- §34.1.3 [Using conditional text to differentiate output](#) on page 936

34.2 Converting a single chapter of a book

When you convert a single file in your book after making some edits, always do the conversion as follows:

1. Open the book file.
2. Open the chapter file from the book file.
3. **Save Using Mif2Go...** from the chapter file.

This way you get the `.prj` (**Mif2Go** project file) settings for the book, and the output conversion files are updated automatically. *Never convert a chapter alone without the book file open in FrameMaker.*

If you change only wording in one chapter, just rerun **Mif2Go** on that chapter, and the entire book will be updated accordingly. However, if you add new cross references that link to other chapters (thus modifying those chapters also), or if you make changes to several chapters, it is better to run **Mif2Go** on the entire book.

34.3 Considering ways to automate conversions

To automate **Mif2Go** conversions, you can do either or both of the following:

- from outside FrameMaker, run one or more conversion projects unattended; either:
 - set up FrameMaker to run from the command line, invoke **Mif2Go** automatically, and optionally close when all conversions are finished; see §36 [Converting via runfm](#) on page 979
 - use the **Mif2Go** DCL filter at a command prompt to convert MIF files; see §37 [Converting via DCL](#) on page 995.
- within FrameMaker, have **Mif2Go** do any or all of the following:
 - assemble files, compile Help systems, and archive deliverables; see §35 [Producing deliverable results](#) on page 955
 - execute system commands before the conversion, before compiling and archiving, at the end of archiving, or all three; see §34.4 [Executing operating-system commands](#) on page 937
 - prompt you for run-time values for user variables (except, of course, during unattended operation); see §34.5 [Supplying run-time values for user variables](#) on page 941.

See also:

§35 [Producing deliverable results](#) on page 955

§36 [Converting via runfm](#) on page 979

§37 [Converting via DCL](#) on page 995

34.4 Executing operating-system commands

Suppose you always check files out of a source-control system before you convert them, and check them back in afterward; or suppose you always copy generated files to multiple locations after conversion. **Mif2Go** can perform these kinds of chores automatically by executing operating-system commands that you specify in the project configuration file.

Note: When you use system commands, exit and restart FrameMaker between projects; otherwise, some internal variables might not be re-initialized.

In this section:

§34.4.1 [Specifying system commands](#) on page 938

§34.4.3 [Monitoring system command execution](#) on page 939

§34.4.4 [Changing configuration settings with system commands](#) on page 940

§34.4.5 [Supplying system commands in a .bat file](#) on page 940

§34.4.6 [Supplying system commands in a macro](#) on page 940

34.4.1 Specifying system commands

To specify a command (or a macro) to execute before or after a document is converted (and optionally compiled and/or archived):

```
[Automation]
; SystemStartCommand = command line to run at start of processing
; SystemWrapCommand = command line to run at end, before compiling
;   and archiving
; SystemEndCommand = command line to run at end, after compiling
;   and archiving
```

Use only commands that can run without interaction.

The value you assign to one of the System*Command keywords is an actual Windows system command, just as you would have typed it at a Windows command prompt. For example:

```
[Automation]
SystemEndCommand = copy /Y G:\MyProj\_wrap\*.xml D:\xml\backups
```

If you specify a relative path in a system command, that path is considered to be relative to the project directory. For example, the following command renames a file located in the wrap directory (see §35.2 [Activating and logging production of deliverables](#) on page 956):

```
[Automation]
SystemEndCommand = rename .\wrap\ugmif2go.htm _ugmif2go.htm
```

Assign only one command to each keyword; the command must be all one line. If you need multiple commands or multiple lines per keyword, see the following:

§34.4.5 [Supplying system commands in a .bat file](#) on page 940

§34.4.6 [Supplying system commands in a macro](#) on page 940.

When you assign a system command (or a macro) to a System*Command, **Mif2Go** generates one or more lines of code, each of which is a command to be run at a Windows command prompt. **Mif2Go** writes these lines to a .bat file named for the keyword, saves the file in your project directory, and causes Windows to execute the file.

*Use backslashes
in file paths*

When you specify a file path in a system command, use “\” as the separator character. For example, suppose you want to make a backup copy of your book on another server before you run each conversion, and then copy your result files to another directory:

```
[Automation]
SystemStartCommand = copy <$$_prjpath>\*.fm x:\backup
SystemEndCommand = copy <$$_currpath>\*.htm \outcopy
```

*Start commands
work only when
you convert*

If your configuration file includes the following setting:

```
[Automation]
OnlyAuto = Yes
```

commands assigned to SystemWrapCommand and to SystemEndCommand are executed; however, commands assigned to SystemStartCommand are ignored. When you set OnlyAuto=Yes, you are deploying an existing set of output files, and you do not want that set disturbed; see §35.13 [Postprocessing separately from converting](#) on page 976.

34.4.2 Including macros and variables in system commands

System commands can include the following:

- **Mif2Go** macro expressions
- user variables you have defined in [UserVars]
- macro variables you have defined in [MacroVariables]
- the following predefined macro variables (see §28.3.4 [Using predefined macro variables](#) on page 800):

<\$\$_basename>	Base file name (without path or extension) of the current FrameMaker source file
<\$\$_currpath>	Path (without trailing slash) to the current directory where the configuration file resides
<\$\$_macroparam>	Value of a parameter passed to the enclosing macro.
<\$\$_prjpath>	Path (without trailing slash) to the directory where the .prj file resides

Predefined macro variables other than those listed here do not work in system commands. Macro variable <\$\$_macroparam> can be used only within a macro; see §28.7 [Passing a parameter to a macro](#) on page 820.

Include macro expressions

You can use macro expressions in system commands: math and string manipulations, conditional expressions, loops, formatted output, and so forth; see §28.6 [Using expressions in macros](#) on page 811.

Prompt for user variables

You can have **Mif2Go** prompt you for values of user variables that appear in system commands; see §34.5 [Supplying run-time values for user variables](#) on page 941.

34.4.3 Monitoring system command execution

To make system commands (and Windows system responses) visible in a command-prompt window while a conversion is running:

```
[Automation]
; SystemCommandWindow =
; Hide (default, no display),
; Show (show during execution only),
; Keep (show until user dismisses)
SystemCommandWindow = Show
```

When you specify Show or Keep for SystemCommandWindow, the system-command .bat file starts with the following lines:

```
REM For: path\to\sourcefilename
@ECHO Running batfilename
@ECHO ON
```

The @ECHO ON command causes the rest of the commands in the .bat file to be visible as they are executed; however, the display might be very brief unless you have a huge project. If there is an error, the error message displays even before the Running batfilename line, an unavoidable Windows feature (because stderr cannot be redirected to stdout).

The next line in the .bat file after your system command is:

```
@ECHO Finished batfilename
```

If SystemCommandWindow=Keep, the .bat file ends with:

```
@PAUSE
```

so that you can see what happened.

Note: Do not specify `SystemCommandWindow=Keep` for unattended use, because the .bat file would wait forever for you to press a key.

The .bat file remains in the project directory, so you can see what it contains. The next time you run the same kind of command, **Mif2Go** recycles the .bat file.

34.4.4 Changing configuration settings with system commands

You can use a system command to make an arbitrary run-time change to a project configuration file, by having the command invoke command-line utility `setini.exe`, which is included in the **Mif2Go** distribution.

Command `setini` changes the value of a single setting:

```
setini inifile.ini section keyword value
```

For example:

```
setini _m2html.ini HTMLOptions NoFonts Yes
```

would exclude the use of `` tags in HTML output. To achieve this via system command, you would specify:

```
[Automation]
SystemStartCommand = setini _m2html.ini HTMLOptions NoFonts Yes
```

34.4.5 Supplying system commands in a .bat file

You can use a text editor to create a Windows .bat file, put system commands in that file, and assign the file name (along with any required path and parameters) to a `System*Command` keyword. For example:

```
[Automation]
SystemEndCommand = buildjh 40
```

The file `buildjh.bat` contains a series of commands to build release 40 of a JavaHelp system. See Windows Help for the syntax required for a .bat file to process parameters.

Note: Because system commands in .bat files require Windows command syntax, you cannot use **Mif2Go** variables in .bat files.

34.4.6 Supplying system commands in a macro

You can put system commands in a **Mif2Go** macro. A macro consists of a special configuration-file section to which you give a unique name; you invoke the macro by assigning its name, enclosed in `<$ >`, to a `System*Command` keyword. See §28.1 [Defining and invoking macros](#) on page 787.

For example, suppose your workflow requires backing up your FrameMaker files to two servers. You could define a macro to supply the two copy commands, and assign that macro to a system command:

```
[Automation]
SystemStartCommand = <$backup>

[backup]
copy <$$_currpath>\\*.fm x:\\backup
copy <$$_currpath>\\*.fm "y:\\my other\\backup"
```

Notice the doubled backslashes (required in **Mif2Go** macros, where backslash is used as an escape character), and the quotes around the path that includes a space. See §28.1.1 [Defining macros](#) on page 787.

Prompt for values
of variables in
macros

You can have **Mif2Go** prompt you for a value each time a `System*Command` is processed; see §34.5 [Supplying run-time values for user variables](#) on page 941.

For example, suppose the second back-up location changes frequently. **Mif2Go** could prompt you for a path name each time a back-up command is executed:

```
[Automation]
AskForUserVars=Always
SystemStartCommand=<$backup>

[UserVars]
back2=H:\Docfiles\backup2

[UserVarPrompts]
back2=Enter pathname for backup 2:

[backup]
copy <$_prjpath>\*.fm x:\backup
copy <$_prjpath>\*.fm <$back2>
```

When you start the conversion, **Mif2Go** prompts you for the value of `back2`. Your response replaces the initial default value in the configuration file. Next time you run the conversion, the value you supplied last time appears as the initial value.

Comment out
commands in
macros

To omit running a particular system command without actually deleting the macro line that executes the command, you can “comment out” the command by preceding it with a semicolon. For example, suppose you do not always want to create a second backup:

```
[backup]
copy <$_currpath>\*.fm x:\backup
; copy <$_currpath>\*.fm "y:\my other\backup"
```

34.5 Supplying run-time values for user variables

If some user variables change from one conversion run to the next, or if you do not know what value to specify for a certain user variable until you are about to start a conversion, you can have **Mif2Go** prompt you for a value. The prompt appears in the *Edit User Variable* dialog, shown in [Figure 34-4](#) on page 943.

In this section:

- §34.5.1 [Assigning an initial value to a user variable](#) on page 941
- §34.5.2 [Assigning a prompt to a user variable](#) on page 942
- §34.5.3 [Deciding how often to prompt for values of user variables](#) on page 942
- §34.5.4 [Understanding when Mif2Go prompts for user variables](#) on page 942
- §34.5.5 [Inspecting and editing values of user variables](#) on page 943

34.5.1 Assigning an initial value to a user variable

To be prompted for the value of a user variable, you must assign the variable an initial default value in the following section:

```
[UserVars]
; User variables can be used in book commands, in macros, or both.
; Enclose each variable in <$_...>.
; The [UserVars] section is searched after section [MacroVariables]
; in the .ini file, but before the same section in the macros file.
varname=initial value
```

The value you assign is the value **Mif2Go** displays via the *Edit User Variable* dialog at run time. If you change the value via the *Edit User Variable* dialog, **Mif2Go** stores the new

value in the configuration file in place of the initial default value. See §34.5.5 [Inspecting and editing values of user variables](#) on page 943.

Note: The [UserVar] section must be in your project configuration file, not in a configuration template. See §30.6.2 [Deciding what to include in a general configuration template](#) on page 862

34.5.2 Assigning a prompt to a user variable

To specify what the *Edit User Variable* dialog displays for a prompt, assign a phrase to the variable in question:

```
[UserVarPrompts]
varname=Prompt to use when a value is requested
```

If you do not specify a prompt for a user variable, **Mif2Go** displays the following default prompt:

Edit content of variable, or supply new content:

For example:

```
[UserVarPrompts]
Version=Enter current version number:
```

Note: The [UserVarPrompts] section must be in your project configuration file, not in a configuration template. See §30.6.2 [Deciding what to include in a general configuration template](#) on page 862

34.5.3 Deciding how often to prompt for values of user variables

To tell **Mif2Go** that you want to be prompted for values of user variables, you must specify when to display the prompt:

```
[Automation]
; AskForUserVars = Never (default), Always, or Once
```

Settings for AskForUserVars have the following effects:

Never	You are not prompted for values of user variables; Mif2Go uses whatever values are specified in [UserVars].
Always	You are prompted for values each time you run the conversion; any value you supply becomes the new default value in [UserVars].
Once	You are prompted for values the first time you run the conversion; then Mif2Go resets AskForUserVars to Never, so the next time you run the same conversion, the default values of user variables are whatever you specified on the first run (unless, of course, you change the setting for AskForUserVars back to Always or Once).

See §34.5.1 [Assigning an initial value to a user variable](#) on page 941.

34.5.4 Understanding when Mif2Go prompts for user variables

Mif2Go prompts for user-variable values at the start of a conversion. **Mif2Go** reads the main configuration file first, and then prompts for values of variables:

- before reading file-specific configuration files, if you are converting a book
- after reading any file-specific configuration file, if you are converting a single file.

As a result, if you specify an initial value for user variable *MyVar* in *filename.ini*, you are prompted for the value of *MyVar* when you are converting *filename.fm* by itself; but

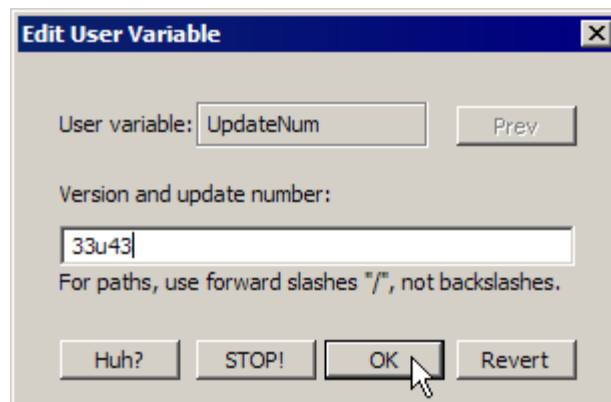
you would be prompted for the value of *MyVar* when you are converting a book that includes *filename.ini* only if you also specified an initial value for *MyVar* in the main configuration file for the book.

34.5.5 Inspecting and editing values of user variables

To prompt you for the values of user variables, **Mif2Go** displays the *Edit User Variable* dialog, shown in [Figure 34-4](#). When the dialog opens, you see the following:

- the name of the first user variable listed in [UserVars]; see §34.5.1 [Assigning an initial value to a user variable](#) on page 941
- the prompt you specified for that variable (or the default prompt); see §34.5.2 [Assigning a prompt to a user variable](#) on page 942
- the default value of the variable (as specified in [UserVars]).

Figure 34-4 *Edit User Variable* dialog



>> To change the value of a variable, edit the value displayed in the text box.

>> To change the variable back to its default value, click **Revert**.

>> To save the value of a variable, click **OK**; **Mif2Go** stores the value in the [UserVars] section of the configuration file. If there is another variable listed in [UserVars], **Mif2Go** displays its name, prompt, and value; if not, the dialog is dismissed.

>> To inspect (or change) the value of a previously displayed variable, click **Prev**. If your pointing device is set to “Snap to default”, proceed cautiously, because each time you click **Prev** the pointer will snap back to the **OK** button.

>> To cancel the conversion, click **STOP!** Any values you already changed are stored in the configuration file as the new defaults.

34.6 Supporting document review in Word

Mif2Go converts FrameMaker documents to Word, but **Mif2Go** does not convert Word documents to FrameMaker (see §6.1 [Converting to Word: a one-way street](#) on page 141). However, in an environment where writers use FrameMaker and everyone else uses Word, you can still create a workable review process in which reviewers get to work in Word, which they already know how to use, and writers maintain the documents in FrameMaker:

1. Create a FrameMaker conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67) to make converted documents look almost the same in Word as they look in FrameMaker.

2. Create a Word template with macros set up to turn on revision tracking automatically whenever anyone opens a converted document.
3. Use **Mif2Go** to convert FrameMaker documents to RTF.
4. Place the converted documents where your reviewers can access them; probably on a network drive.
5. Allow reviewers time to edit the documents in Word. Any changes they make show up in color. Word not only tracks the changes, but also shows who made them, so if you have questions you can contact the author.
6. When reviewers are finished, use Paste Special in FrameMaker for plain-text (*not* RTF) copy/paste, to incorporate Word changes into the original FrameMaker document. Pasting as plain text will save you endless grief from the (mostly invisible) artifacts that are retained in your FrameMaker file when you use RTF import.
7. Use **Mif2Go** to convert the updated FrameMaker documents to RTF again, so reviewers can open the converted documents in Word and see the results.

Do not attempt to “round-trip” an edited Word document back into FrameMaker; in most cases the clean-up time and cost would far exceed the time and cost to insert edits by hand in FrameMaker. Problems might not be visible immediately, but then will surface when you are in final production. For example, applying to the reimported document a template that changes character properties of paragraph formats might appear not to work at all, because a character format was silently added to *all* default-format text. Even when changes are extensive, it is best to use copy and paste as plain text, one FrameMaker paragraph at a time.

See §6 [Converting to print RTF](#) on page 141.

34.7 Converting autonumbers for database systems

Suppose you use FrameMaker autonumbers for headings of the style you see in the **Mif2Go User's Guide**; and suppose you use an automated system to populate a database with the number and text of each heading, from **Mif2Go**-generated HTML output. You could use macros and macro variables to capture the numerical value of each autonumber, and perhaps output the number as the name value of a tag, such as ``.

For example, suppose in FrameMaker you have three heading format levels, with the following autonumber scheme:

<i>Chapter</i>	H: <\$chapnum> < =0>< =0>
<i>Heading1</i>	H: <\$chapnum> .<n+> < =0>
<i>Heading2</i>	H: <\$chapnum> .<n> .<n+>

Suppose all the headings are bold, including the autonumbers. In HTML output these heading formats might look like the following:

2 This is a chapter title
13.5 This is a second-level heading
6.2.7 This is a third-level heading

To capture each autonumber as a six-digit number, with a leading zero (as needed) for each level (for example, 060207), you could provide settings and macros such as the following:

```
[HTMLParaStyles]
Chapter=Split Title CodeStore CodeAfter
```

```

Heading1=Split Title CodeStore CodeAfter
Heading2=Split Title CodeStore CodeAfter

[StyleCodeStore]
; Set aside in a macro variable the code generated for each heading:
*=Stored

[ParaStyleCodeAfter]
; Parse the autonumber of each heading format; insert the resulting
; six-digit number as <a name=nnnnnn>, and then output the stored
; heading itself:
Chapter=<$ParseAnum><a name="<$ChapNum>"><$$Stored></a>
Heading1=<$ParseAnum><a name="<$Hdg1Num>"><$$Stored></a>
Heading2=<$ParseAnum><a name="<$Hdg2Num>"><$$Stored></a>

[ChapNum]
; Chapter number followed by four zeros:
<$Chap as %0.2d>0000\

[Hdg1Num]
; Chapter number, then Heading1 number, then two zeros:
<$Chap as %0.2d><$$Hdg1 as %0.2d>00\

[Hdg2Num]
; Chapter number, then Heading1 number, then Heading2 number:
<$Chap as %0.2d><$$Hdg1 as %0.2d><$$Hdg2 as %0.2d>\

[ParseAnum]
; Pick through the stored code to pull out successive pieces of
; the autonumber, and put them in separate macro variables:
<$$Text = ($$Stored after "<b>")>\
<$$Anum = ($$Text before " ")>\
<$$Chap = ($$Anum before ".")>\
<$$Anum2 = ($$Anum after ".")>\
<$$Hdg1 = ($$Anum2 before ".")>\
<$$Hdg2 = ($$Anum2 after ".")>\

```

Trailing backslashes in the macro code prevent hard line breaks from going into the HTML output.

As each heading is processed, **Mif2Go** sets aside the generated HTML code in macro variable `$$Stored`. **Mif2Go** parses the stored code as follows, to extract the autonumber:

1. Skips everything in the stored code up through the `` tag.
2. Puts in `$$Anum` everything between the `` tag and the next space; this includes the whole autonumber.
3. Stores in `$$Chap` the characters before the first period in `$$Anum`.
4. Puts in `$$Anum2` the characters after the first period in `$$Anum`.
5. Stores in `$$Hdg1` the characters before the first period in `$$Anum2`.
6. Stores in `$$Hdg2` the characters after the first period in `$$Anum2`.
7. Assembles each six-digit number. The `%0.2d` format specifier takes care of any dangling tags, and provides any needed leading zeros.

Back in the individual format settings in `[ParaStyleCodeAfter]`, **Mif2Go** puts out the start of the `<a>` tag, and then whichever of the macro variables is needed. Finally, **Mif2Go** adds the original stored heading itself to the output, and closes the `<a>` tag.

See §28 [Working with macros](#) on page 787.

34.8 Renaming output files for automated systems

For names of **Mif2Go**-generated files, **Mif2Go** is more restrictive than Windows. Only alphanumeric characters and upper-ASCII accented characters are allowed; no punctuation at all (except a leading underscore for starting topic files), and no spaces, unless you explicitly override these restrictions. See §1.1.2 [File, directory, and path names](#) on page 51.

The ability to respecify output file names is available for a single purpose: to allow creation of files that external software tools need to have named a particular way. Doing so works well for that purpose. **Do not try to rename split files unless you are constructing an automated system.** Why? Because our experience shows there is a very high probability of name collisions.

Note: Renaming an output file *outside* of **Mif2Go** breaks any links to that file; however, see §19.6.3 [Enabling links to renamed or relocated files](#) on page 622.

In this section:

§34.8.1 [Understanding which files can be renamed](#) on page 946

§34.8.2 [Renaming individual output files](#) on page 946

§34.8.3 [Using custom markers to name output files](#) on page 947

§34.8.4 [Using paragraph formats to name output files](#) on page 947

§34.8.5 [Including identifiers and sequence numbers in file names](#) on page 952

34.8.1 Understanding which files can be renamed

You can rename only split and extracted HTML and XML files. *You cannot rename the file produced before the first split*; that file must have the same name as its FrameMaker file. For example, suppose you are converting a FrameMaker file named `models.fm`, and splitting it into multiple HTML files. The first file created, which will most likely be empty of content, is named `models.htm`. **Mif2Go** generates the names of the rest of the files.

The reason for this requirement is that when **Mif2Go** checks to see if a referencing (or referenced) file has been created, **Mif2Go** knows only the name of the FrameMaker file, not the names of split files. So **Mif2Go** looks for an HTML file with the same name; if it is not there, **Mif2Go** concludes that the file has not been processed yet, and therefore does not try to update any references it contains to the current file. When **Mif2Go** does see the originally named file, updating can proceed.

You can have **Mif2Go** split each FrameMaker file on the very first heading (see §18.2.2.2 [Preventing splits that create unwanted files](#) on page 588), so that the first output file is empty; then you can leave the empty files behind when you move output files to their final destination. (Do not delete the empty files; doing so could break interfile links. See §20.4.8 [Specifying an alternate file sequence for browse links](#) on page 644.)

34.8.2 Renaming individual output files

*Do not try to rename **Mif2Go**-generated files outside of **Mif2Go**.*

To substitute a different name for a particular split or extract output file, map the original name to the new name (without extension):

```
[HtmlFiles]
; original html filename = desired html filename, only for split
; or extract files, not usable for files named for the original
```



```
; Frame file
splitfilename = newname
```

We strongly advise all-lowercase file names, so that they work from a UNIX server, where references are case sensitive. Do not include spaces or non-alphanumeric characters in file names.

You cannot use the [HtmlFiles] section to rename files other than those produced by splitting; not even the file before the first split point, which retains the original FrameMaker name. **Mif2Go** writes this file even if it is essentially empty.

See also:

§18.2.2.2 [Preventing splits that create unwanted files](#) on page 588.

§18.4.1 [Understanding how split and extract files are named](#) on page 593.

34.8.3 Using custom markers to name output files

To specify a file name via marker, you can create a FrameMaker marker type called **FileName** and insert a **FileName** marker in the first paragraph of each part to be split or extracted. Make the content of each **FileName** marker the name you want for the resulting file, without path or extension.

Duplicated file names are hard to locate

Using **FileName** marker can result in two sections of your document having the same name. When this happens the second file overwrites the first, and the first topic does not appear in output. This error is almost impossible to find, unless you search very specifically through all file-name settings and markers. Expect many problems of this type if you use **FileName** markers to override **Mif2Go**-generated file names.

Use FileName markers only for split or extracted files

FileName markers work for any split or extracted file, but not for the first file produced from each FrameMaker file, which must have the same base name as the FrameMaker file. However, you can specify a split at the very first paragraph in each FrameMaker file, usually a chapter heading. For example:

```
[HTMLOptions]
StartingSplit=Yes

[HTMLParaStyles]
ChapTitle=Split
```

Keep empty files

The FrameMaker-named split files will be empty, and need not be included with your deliverables. However, *do not delete these files from the project directory*; subsequent conversions might require their presence.

As an alternative, you can add the **FileName** property to another marker: for example, custom marker type **Split** (see §18.2.1 [Designating split points](#) on page 586), or custom marker type **ExtrStart** (see §18.7.1 [Using markers for extract processing](#) on page 602); and specify the name of the file as the marker content.

See also:

§29 [Working with FrameMaker markers](#) on page 831

§18 [Splitting and extracting files](#) on page 585

34.8.4 Using paragraph formats to name output files

According to **Mif2Go** developers, naming output files using paragraph content is *a Very Bad Idea*. You are almost certain to have name conflicts that result in **Mif2Go** overwriting one file with another, and you will not know it happened until users complain.

However, at your peril, you can assign file names based on the content of paragraphs: either existing paragraphs (usually heading paragraphs whose formats designate split points), or paragraphs in a special format that you dedicate to this purpose.

To help ensure uniqueness of file names, you can also specify a fixed or variable file-name prefix or suffix, or both.

In this section:

- §34.8.4.1 [Constructing file names based on paragraph content](#) on page 948
- §34.8.4.2 [Including FrameMaker variables in output file names](#) on page 949
- §34.8.4.3 [Basing output file names on existing paragraph formats](#) on page 949
- §34.8.4.4 [Creating special paragraph formats to name output files](#) on page 950
- §34.8.4.5 [Specifying a file-name prefix or suffix](#) on page 950
- §34.8.4.6 [Constructing file names from multiple paragraph formats](#) on page 951
- §34.8.4.7 [Preventing duplicate file names based on paragraph formats](#) on page 952

34.8.4.1 Constructing file names based on paragraph content

You can specify names for HTML or XML output files by designating a FrameMaker paragraph format to use for this purpose, and listing the format name in the [HTMLParaStyles] section. The content of each paragraph in this format becomes the base name of a new split part:

- prefixed with whatever you specify for [StyleFilePrefix],
- suffixed with whatever you specify for [StyleFileSuffix], and then
- followed by the file extension.

To use a paragraph format to name split files, assign the FileName property to the format:

```
[HTMLParaStyles]
paratag = FileName
```

*Object ID
replaces
unusable content*

If the content of a paragraph to which you assign the FileName property is empty, or consists only of characters that are not valid for file names, **Mif2Go** uses the ObjectID of the paragraph for the file name instead (see §18.4.1 [Understanding how split and extract files are named](#) on page 593), along with any prefix or suffix you specify for file names (see §34.8.4.5 [Specifying a file-name prefix or suffix](#) on page 950).

*Ensure valid file
names*

These cobbled-together split-file names are guaranteed to consist of valid file-name characters *only* with the following default setting:

```
[HTMLOptions]
; UseRawName = No (default, make [HTMLParaStyles] FileName valid)
; or Yes
UseRawName = No
```

When UseRawName=Yes, file names generated from paragraphs retain the full content of the paragraph, including any whitespace and punctuation; that is, unless the paragraph consists only of non-alphanumeric characters, in which case **Mif2Go** uses the ObjectID of the paragraph for the file name.

When UseRawName=No, all whitespace and punctuation are removed from the file name, unless you set either or both of the following options to Yes; *if you set either option, we can no longer guarantee that the generated file names will be valid:*

```
[HTMLOptions]
; When UseRawName=No, allow underscores and spaces to be passed
; through from headings with the FileName property as follows:
; KeepFileNameUnderscores = No (default, remove underscores) or Yes
KeepFileNameUnderscores = Yes
```

```
; KeepFileNameSpaces = No (default, remove or change spaces) or Yes
KeepFileNameSpaces = Yes
```

When `KeepFileNameSpaces=No`, you can choose to replace each space in the file name with some other character:

```
[HTMLOptions]
KeepFileNameSpaces = No
; ChangeFileNameSpaces = No (default; if not kept, remove) or
; Yes (if not kept, replace with the FileNameSpaceChar, below)
ChangeFileNameSpaces = Yes
; FileNameSpaceChar = character with which to replace spaces,
; default '_', used if both KeepFileNameSpaces=No and
; ChangeFileNameSpaces=Yes
FileNameSpaceChar = _
```

The default replacement character is an underscore. The setting for `FileNameSpaceChar` takes effect only if both of the following are true:

- `KeepFileNameSpaces = No`
- `ChangeFileNameSpaces = Yes`.

The only non-alphanumeric character replaced is the space. All other non-alphanumeric characters are removed. For example:

```
Basic 40/41/42 Chipset
```

becomes:

```
Basic_404142_Chipset
```

The forward slashes are removed.

34.8.4.2 Including FrameMaker variables in output file names

To construct HTML or XML output file names based in part on the values of FrameMaker variables present in the document you are converting, you might have to employ additional **Mif2Go** macros to make variable values comply with the requirements for **Mif2Go** file names. This is because the following options described in §34.8.4.1 [Constructing file names based on paragraph content](#) on page 948 *have no effect* on content derived from FrameMaker user variables:

```
UseRawName
KeepFileNameSpaces
ChangeFileNameSpaces
FileNameSpaceChar
```

For example, to replace spaces in the content of a FrameMaker user variable, you might have to use a macro such as the following to transform the variable value to a string that is acceptable for inclusion in a file name:

```
<$$newval = ($$myspecialvar replace " " with "-")>
```

See also:

§28.3.5 [Treating FrameMaker user variables as macro variables](#) on page 801

§28.3.6 [Using some FrameMaker system variables as macro variables](#) on page 802

§28.6.5 [Specifying substrings in expressions](#) on page 817

§34.8.4.5 [Specifying a file-name prefix or suffix](#) on page 950

34.8.4.3 Basing output file names on existing paragraph formats

If you use an existing paragraph format (usually a heading) in your FrameMaker document to mark split points, and the paragraphs in this format already contain

appropriate text for output file names, you can assign the `FileName` property to that format, and if necessary specify prefix and suffix (see §34.8.4.5 [Specifying a file-name prefix or suffix](#) on page 950). This is a simple way to use titles for file names.

However, if you are creating HTML Help (see §9 [Generating Microsoft HTML Help](#) on page 295) you would be asking for trouble. Most Help systems have files with identical titles at several points; titles such as “Summary” or “Overview” often appear under several topics, so using the title as the file name is almost certain to cause name collisions, unless you also include a unique identifier in the prefix or suffix, such as the `FileID` and a sequence number. See §34.8.5 [Including identifiers and sequence numbers in file names](#) on page 952.

If you ever duplicate a `FileName` heading in the same file, you are in deep trouble with no warning. The later file will silently overwrite the earlier. *It is your responsibility to detect and avoid potential collisions*, by changing the text of duplicate headings, or insuring uniqueness via sequence numbers. See §34.8.4.7 [Preventing duplicate file names based on paragraph formats](#) on page 952 for another way to accomplish this. In a large Help system, you might have to use a DBMS (Data Base Management System), such as SQL Server or Access, for the names.

34.8.4.4 Creating special paragraph formats to name output files

A way to assign file names that is slightly less hazardous than using titles, but still unsafe, is to specify a special paragraph format to hold the names. If paragraphs in this format are used solely for naming files, most likely you do not want them to actually appear in the output. To prevent their appearance, specify the `Delete` property:

```
[HTMLParaStyles]
ParaFmt = FileName Delete
```

Insert a new paragraph with format `ParaFmt` anywhere after a split heading and before the next split point. Although the element paragraph whose format you designated can be anywhere in the split file, usually you would put it right after the heading that starts the split. **Mif2Go** uses the content of that paragraph as the base part of the file name. The `Delete` property removes the paragraph from the HTML output (see §21.3.12 [Eliminating unwanted paragraphs](#) on page 652); you can use conditional text to remove it from your FrameMaker print version.

For example, to supply your own names for all the HTML topic files to be generated from your FrameMaker document, you could do the following:

1. Create and catalog a FrameMaker paragraph format (for example, *Splitname*).
2. Place a *Splitname* paragraph in each portion of your FrameMaker document that will become a separate HTML topic file.
3. Make the contents of each *Splitname* paragraph the base file name you want for that particular topic.
4. Make all *Splitname* paragraphs conditional, so they do not show up in print versions of your FrameMaker document.
5. Specify the following setting:

```
[HTMLParaStyles]
Splitname=FileName Delete
```

34.8.4.5 Specifying a file-name prefix or suffix

You can specify a prefix, a suffix, or both, for format-based names of HTML output files:

```
[StyleFilePrefix]
; doc format = prefix to use (if any) for file name in para content
parafmt=splitfileprefix

[StyleFileSuffix]
; doc format = suffix to use (if any) for file name in para content
parafmt=splitfilesuffix
```

If you are splitting files at *Heading1* paragraphs, for example, you could specify the following properties:

```
[HTMLParaStyles]
Heading1=Split Title FileName

[StyleFilePrefix]
Heading1=ug

[StyleFileSuffix]
Heading1=03
```

If a given instance of *Heading1* consists of the text “Getting Started”, the resulting HTML filename would be `ugGettingStarted03.htm`.

You can use macros and macro variables (see §28.1 [Defining and invoking macros](#) on page 787) in sections `[StyleFilePrefix]` and `[StyleFileSuffix]`.

For example, suppose you have defined a *FrameMaker* variable called *BkNum* in your document, and you want to use the value of that variable as a file-name prefix:

```
[StyleFilePrefix]
Heading1=<$$BkNum>
```

This works because *FrameMaker* user variables can be employed as **Mif2Go** user variables; see §28.3.1 [Creating and invoking macro variables](#) on page 796. To make the content conform to file-name requirements, see §34.8.4.2 [Including FrameMaker variables in output file names](#) on page 949.

You can also use predefined macro variables; for example, `<$$_objectid>`, which has the advantage of guaranteeing that each file name will be unique. See §29.8 [Identifying markers with variable <\\$\\$_objectid>](#) on page 847.

34.8.4.6 Constructing file names from multiple paragraph formats

Suppose you split files on both *Heading1* and *Heading2* paragraph formats, and you want each *Heading2* split-file name to be prefixed by the content of the preceding *Heading1* paragraph. You can use the `TextStore` property to capture the content of each succeeding *Heading1* paragraph, and make that content available to all *Heading2* split files up to the next *Heading1* paragraph:

```
[HTMLParaStyles]
ChapterTitle = Split Title FileName
Heading1 = Split Title FileName TextStore
Heading2 = Split Title Filename

[StyleFilePrefix]
Heading2 = <$($$Heading1 replace " " with "_")>_
```

The `TextStore` property uses the format name by default for the name of the variable it creates (see §28.3.7.1 [Capturing paragraph content with the TextStore property](#) on page 803), so you can simply specify `$$Heading1` in the prefix value. You can use a macro expression to replace any spaces in *Heading1* content; see §28.6.5 [Specifying substrings in expressions](#) on page 817.

34.8.4.7 Preventing duplicate file names based on paragraph formats

To ensure uniqueness of file names without using a prefix or suffix, you can have **Mif2Go** override a file name, by combining the two paragraph-based file-naming methods:

1. Add a special paragraph format in FrameMaker to enable creation of a file name other than from an existing paragraph, as described in §34.8.4.4 [Creating special paragraph formats to name output files](#) on page 950.
2. Use the `FileName` property in `[HTMLParaStyles]` for both the existing paragraph and the special paragraph formats. The file-name paragraph should *follow* (not precede) the heading paragraph; the *last* name specified wins.

For example:

```
[HTMLParaStyles]
Heading1 = Split Title FileName
Splitname = FileName Delete
```

34.8.5 Including identifiers and sequence numbers in file names

To ensure uniqueness of file names for split files, when you are constructing file names based on paragraph formats you might want to include in the file-name prefix or suffix any or all of the following predefined macro variables:

<\$\$_fileid>	FileID (as specified in <code>mif2go.ini</code>) of the FrameMaker file from which the file was split
<\$\$_splitid>	Base name of the split file, excluding the FileID portion
<\$\$_splitnum>	A sequential number whose starting value and increment you can specify

To specify a starting value (default 0) and increment (default 1) for <\$\$_splitnum>:

```
[HtmlOptions]
SplitNumStart=0
SplitNumIncrement=1
```

The value of <\$\$_splitnum>:

- begins at `SplitNumStart` for the first split file
- increments by `SplitNumIncrement` for each subsequent split file
- starts over again at `SplitNumStart` for each FrameMaker file.

You can format <\$\$_splitnum> to include leading zeroes, and to specify the number of digits; see §28.6.3 [Displaying expression results in output](#) on page 813. For example:

```
[StyleFilePrefix]
Head*=<$$_fileid><$$_splitnum as %0.3d>
```

You could use predefined macro variable <\$\$_basefile> instead of (or in addition to) <\$\$_fileid>, to supply in a prefix or suffix the base name of the FrameMaker file from which each HTML file was split:

```
[StyleFilePrefix]
Head*=<$$_fileid><$$_basefile><$$_splitnum as %0.3d>
```

You can include the base file name that **Mif2Go** assigns to the split file. For example:

```
[HTMLParaStyles]
Head*=Split Title FileName

[StyleFilePrefix]
Head*=<$$_basefile>_<$$_splitnum as %0.4d>_
```

```
[StyleFileSuffix]  
Head*=_<$$_fileid><$$_splitid>
```

*Sequence
numbers have
their drawbacks*

There are good reasons *not* to use sequence numbers in file names. With sequence numbers, almost any revision can result in changes to file names. If you insert a new topic, every topic that follows gets a new file name. The default **Mif2Go** file naming method ensures the exact opposite: that hardly any revisions result in a file name change; the most you usually get is a name addition for an added topic. **Mif2Go** does *not* change the file names for all the following topics when you insert a new topic. Why is this important? Because any links from other documents and projects, possibly created by other writers, will be broken unless you rebuild absolutely everything in the document system, which could be thousands of files. With the default method, nothing will break. If you think you are the only one who cares about those links, do you know that will always be the case?

35 Producing deliverable results

Mif2Go can automatically handle a certain amount of pre- and post-conversion processing to prepare deliverables. This section describes the steps you can automate. Topics include:

- §35.1 [Understanding Mif2Go pre- and post-processing](#) on page 955
- §35.2 [Activating and logging production of deliverables](#) on page 956
- §35.3 [Understanding path values for deliverables](#) on page 957
- §35.4 [Clearing out old files before converting](#) on page 957
- §35.5 [Gathering additional files before converting](#) on page 960
- §35.6 [Assembling files for distribution](#) on page 961
- §35.7 [Placing graphics files for distribution](#) on page 965
- §35.8 [Placing CSS or XSL files for assembly](#) on page 969
- §35.9 [Gathering files for an HTML project: an example](#) on page 970
- §35.10 [Gathering and processing Help-system files](#) on page 971
- §35.11 [Archiving deliverables](#) on page 973
- §35.12 [Placing deliverables in a shipping directory](#) on page 975
- §35.13 [Postprocessing separately from converting](#) on page 976

See also:

- §34 [Automating Mif2Go conversions](#) on page 933

35.1 Understanding Mif2Go pre- and post-processing

When you convert a document, **Mif2Go** usually places all the files generated in the course of the conversion in the project directory. As a result, the project directory subsequently contains not only newly converted document files, but also configuration files and generated conversion files that are not part of the converted document. It might even contain obsolete output files from a previous conversion. See §C [Document and conversion files](#) on page 1019.

For many output types, when you prepare a converted document for distribution you need to separate the wheat from the chaff. It is a good idea to copy the converted files, along with any other files that must be distributed with the output, to a directory where they can be accessed by others, or easily compiled or archived for distribution. In many cases **Mif2Go** can handle the compiling or archiving for you.

Before generating output files, **Mif2Go** can do the following:

- Delete prior MIF files from the project directory. If you have just edited and regenerated your FrameMaker documents, you do not need the MIF files from a prior conversion. Deleting them instead of overwriting them avoids creating FrameMaker back-up files, and speeds up conversion.
- Delete prior output and conversion files from the project directory. Best not to leave orphaned and obsolete files where they can be swept up into a new distribution.
- Copy needed files into the project directory, such as configuration files and CSS files that you keep in a central, safe location.

After generating output files, **Mif2Go** can do any or all of the following:

- Assemble files for distribution:
 - Create a separate directory (or a directory structure) where results of a conversion, along with ancillary files such as graphics, can be assembled for compiling,

- archiving, distribution, or use. Or, use an existing directory (or directory structure) you designate for this purpose.
- Gather necessary files into the wrap directory (and subdirectories, if appropriate).
- Compile or archive deliverables, or both:
 - Create a separate “shipping” directory for compiled or archived results, or use an existing directory you designate for this purpose.
 - Run a full-text-search indexing program (JavaHelp; putatively, Oracle Help for Java).
 - Run a compiler (WinHelp or HTML Help).
 - Run an archiving program, and place the results in the shipping directory.
- Log any operating-system commands executed in the course of assembling, compiling, and archiving.

If your workflow is more involved than this, you can specify other pre- and post-processing custom steps and arrange for interactive prompts, via system commands; see §34 [Automating Mif2Go conversions](#) on page 933.

35.2 Activating and logging production of deliverables

To have **Mif2Go** assemble files and optionally archive deliverables, either check **Wrap and Ship** on the *Export* dialog, or specify the following option in your project configuration file:

```
[Automation]
; WrapAndShip = No (default) or Yes (use WrapPath, ArchiveCommand,
; ShipPath, CopyGraphicsFrom, and CopyCssFrom)
WrapAndShip=Yes
```

When `WrapAndShip=Yes`, **Mif2Go** acts on all options in the [Automation] section of the configuration file. To have **Mif2Go** place deliverables in a shipping directory, you must also specify a value for `ArchiveCommand`; see §35.11 [Archiving deliverables](#) on page 973.

When `WrapAndShip=No` (the default), **Mif2Go** ignores most [Automation] settings, unless you have **Mif2Go** run a compiler or indexer for a Help system (see §35.10 [Gathering and processing Help-system files](#) on page 971). [Table 35-4](#) on page 972 shows which [Automation] settings are activated when **Mif2Go** runs a compiler or indexer for a Help system.

Note: If you specify `CopyOriginalGraphics=Yes`, graphics are copied regardless of the value of `WrapAndShip`; see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.

*Log the actions
taken*

To have **Mif2Go** log the commands executed when `WrapAndShip=Yes`:

```
[Automation]
WrapAndShip=Yes
; LogAuto=No (default) or Yes (log all automation commands)
LogAuto=Yes
```

When `LogAuto=Yes`, each command **Mif2Go** executes to carry out an automation option is recorded in the **Mif2Go** log file, provided logging is enabled; see §5.2 [Logging conversion events](#) on page 115. `LogAuto` takes effect only when `WrapAndShip=Yes`.

*Log runfm
diagnostic
messages*

If you plan to use **runfm** for unattended production runs (see §36 [Converting via runfm](#) on page 979), while you work out the process you might want to log additional diagnostic messages to the FrameMaker console window:

```
[Automation]
; RunfmDiagnostics = No (default) or Yes (include more diagnostic
; messages in the Frame console file when running from runfm)
RunfmDiagnostics=Yes
```

RunfmDiagnostics takes effect only when WrapAndShip=Yes. When you use **runfm**, you can also capture FrameMaker console messages in a log file. See:

§36.5.1 [Increasing console diagnostics: runfm -diag option](#) on page 988

§36.5.2 [Capturing console diagnostics: runfm -log option](#) on page 988.

35.3 Understanding path values for deliverables

When you set up a new conversion project (see §3.4 [Choosing project set-up options](#) on page 79), **Mif2Go** includes the following settings in your new configuration file:

```
[Automation]
WrapAndShip = Yes
WrapPath = .\_wrap
ShipPath = ..\_ship
```

The default path for WrapPath is relative to the project directory for your project, and the default path for ShipPath is relative to the WrapPath value; therefore, by default:

- `_wrap` becomes a subdirectory of the project directory
- `_ship` becomes a directory parallel to the project directory.

For example, if your FrameMaker files are in `d:\mydoc`, and you specify `d:\mydoc\myout` as the project directory, the default values of WrapPath and ShipPath would specify, respectively, the following locations:

```
d:\mydoc\myout\_wrap
d:\mydoc\_ship
```

You can specify other names and locations for these directories. See:

§35.6.1 [Specifying a wrap directory](#) on page 961

§35.12.1 [Specifying a shipping directory for deliverables](#) on page 975

See also:

§4.5 [Specifying file paths in configuration settings](#) on page 105

35.4 Clearing out old files before converting

By the time you are ready to start a production run, typically you have already completed one or more trial conversions, perhaps leaving MIF files, conversion files, and old output files in the project directory. You can have **Mif2Go** remove these files before starting the next conversion, so they do not slow down the process, and so obsolete and unneeded files do not accidentally end up in a deliverable.

In this section:

§35.4.1 [Specifying when to delete old files from the project directory](#) on page 958

§35.4.2 [Specifying which files to delete from the project directory](#) on page 958

§35.4.3 [Understanding when not to delete .ref and .htm files](#) on page 959

§35.4.4 [Deleting MIF files from the project directory](#) on page 960

See also:

§35.6.2 [Emptying the wrap directory before copying](#) on page 962

35.4.1 Specifying when to delete old files from the project directory

To specify under which conditions certain old files should be deleted from the project directory before conversion:

```
[Automation]
WrapAndShip=Yes
; EmptyOutputDir = Never (default), Book (when running a full book),
; or File (for running a single-file project with no external links).
EmptyOutputDir = Never
```

Values of EmptyOutputDir have the following effects:

- Never *Default.* **Mif2Go** does not delete any files from the project directory before conversion.
- Book **Mif2Go** deletes files before conversion only if you are converting a FrameMaker book. Use this setting when you convert an entire book, or reconvert a single chapter of a book.
- File **Mif2Go** deletes the specified files before conversion only if you are converting a single FrameMaker file. This setting is hazardous; use it *only* for single-file projects with no external links.

To determine which files to delete before conversion, see:

§35.4.2 [Specifying which files to delete from the project directory](#) on page 958

§35.4.3 [Understanding when not to delete .ref and .htm files](#) on page 959.

Note: EmptyOutputDir takes effect not only when WrapAndShip=Yes, but also when one of the following is true for the output type specified:

HTML Help: [Automation]CompileHelp=Yes

WinHelp: [Automation]CompileHelp=Yes

JavaHelp: [JavaHelpOptions]FTSCommand=path/to/indexer

Oracle Help: [OracleHelpOptions]FTSCommand=path/to/indexer

See §35.10 [Gathering and processing Help-system files](#) on page 971.

35.4.2 Specifying which files to delete from the project directory

To specify which files **Mif2Go** should delete from the project directory before conversion:

```
[Automation]
WrapAndShip=Yes
; EmptyOutputFiles = list of files to delete, separated by spaces,
; wildcards allowed but not paths, no spaces within an item
EmptyOutputFiles = *.htm *.ref *.grx
```

If you do not include a setting for EmptyOutputFiles, depending on the value of EmptyOutputDir (see §35.4.1 [Specifying when to delete old files from the project directory](#) on page 958), by default **Mif2Go** deletes the following old files from the project directory before conversion:

<u>Output type</u>	<u>Default files deleted before conversion</u>
HTML (all), XHTML	*.htm *.html *.ref *.grx
XML	*.xml *.ref *.grx
DITA	*.dita *.ditamap *.bookmap .ref .grx
RTF	*.rtf *.grx

Note: If you list either *.dcl or *.dcb for EmptyOutputFiles, **Mif2Go** ignores EmptyOutputDir and logs a warning. To delete .dcl and .dcb files, see §5.1.4 [Reusing or discarding ASCII DCL files](#) on page 111.

Use wildcards; do not use paths

The file specifications you assign to `EmptyOutputFiles` must be separated by spaces, and no spaces are allowed within a file specification. You can use wildcards in file specifications, but you cannot include paths.

Warning: *Do not specify *.* or *.ini, or you will lose your configuration file(s); and for Help systems, you might lose a great deal more.*

Depending on the value of `EmptyOutputDir` (see §35.4.1 [Specifying when to delete old files from the project directory](#) on page 958), **Mif2Go** deletes the specified files before conversion begins, immediately after executing any system commands listed in your configuration file for `SystemStartCommand`, in section `[Automation]`; see §34.4 [Executing operating-system commands](#) on page 937.

Files get deleted on compiling and indexing, also

Note: `EmptyOutputFiles` takes effect not only when `WrapAndShip=Yes`, but also when one of the following is true for the output type specified:

HTML Help: `[Automation]CompileHelp=Yes`

WinHelp: `[Automation]CompileHelp=Yes`

JavaHelp: `[JavaHelpOptions]FTSCommand=path/to/indexer`

Oracle Help: `[OracleHelpOptions]FTSCommand=path/to/indexer`

See §35.10 [Gathering and processing Help-system files](#) on page 971.

See also:

§5.1.4 [Reusing or discarding ASCII DCL files](#) on page 111

§35.4.1 [Specifying when to delete old files from the project directory](#) on page 958

§35.4.3 [Understanding when not to delete .ref and .htm files](#) on page 959

§35.4.4 [Deleting MIF files from the project directory](#) on page 960

35.4.3 Understanding when not to delete .ref and .htm files

If the files you are converting have no interfile links, you do not need `.ref` files. However, if there are interfile links, `.ref` files are essential to make any links to split parts point to the correct split file (see §18.2 [Splitting files](#) on page 586).

If you are converting to any HTML output type, in the following situations you *must* provide an explicit setting for `EmptyOutputFiles` that does *not* include `*.ref`:

- `EmptyOutputDir=Book`, and you are converting a book that has interbook links
- `EmptyOutputDir=File`, and you are converting a file that has interfile links.

External links require keeping .ref files

By default, **Mif2Go** deletes `*.ref` when `EmptyOutputDir` is in play (see §35.4.1 [Specifying when to delete old files from the project directory](#) on page 958) and there is no explicit setting for `EmptyOutputFiles` (see §35.4.2 [Specifying which files to delete from the project directory](#) on page 958). When reference files are removed, you lose the information added from other books or files that were already converted. If you are running a series of such conversions, you can delete `*.ref` files before the first conversion, but not thereafter.

On the other hand, a `.ref` file might actually be inaccurate if it covers a version of a file that has different page breaks; and `.ref` files clutter the project directory. It is best to remove them when you are about to reconvert an entire book.

External links might require keeping .htm files

When you are converting a single file with external links (such as a chapter of a book), for links *from* the file being converted, the `.refs` for the files referenced are needed. For links *to* a file being converted, the `.ref` for that file is needed by the other files. The other `.htm` files are needed too, because the file being converted might patch other `.htm` files when

references shift to a different split part. Therefore, removing .htm files is hazardous when there are interfile references of any sort; broken links can result.

On the other hand, obsolete and orphaned .htm files can end up in a deliverable if they remain in the project directory, so it is best to remove them when you are about to reconvert an entire book.

See also:

§35.4.1 [Specifying when to delete old files from the project directory](#) on page 958

§35.4.2 [Specifying which files to delete from the project directory](#) on page 958

§C.5 [Working with reference files for HTML or XML](#) on page 1027

35.4.4 Deleting MIF files from the project directory

To delete MIF files from the project directory before conversion, see §5.1.3 [Reusing or discarding MIF files](#) on page 111.

As an alternative to `DeleteExistingMIF`, you could list *.mif as one of the file types for `EmptyOutputFiles`. Then, MIF files would be deleted before conversion only when the type of conversion (book vs. single file) is consistent with the setting for `EmptyOutputDir` (see §35.4.1 [Specifying when to delete old files from the project directory](#) on page 958). However, if you expect to rerun a conversion from the book file after manually deleting only those MIF files you want to replace, do not list *.mif as a file type for `EmptyOutputFiles`; all your MIF files will be deleted, regardless of what you check on the *Export* dialog. Use `DeleteExistingMIF` instead.

35.5 Gathering additional files before converting

For safety, or for sharing with other writers, you might keep copies of ancillary files in a location other than the project directory. You can have **Mif2Go** copy those files into the project directory before beginning a conversion.

To copy files into the project directory:

```
[Automation]
; CopyBeforeFrom = path to directory containing files to add to the
; project directory before processing. For example:
CopyBeforeFrom = ..\..\keepers
; CopyBeforeFiles = list of files to copy from CopyBeforeFrom
; to the project directory, separated by spaces, wildcards and
; paths (relative and absolute) allowed, no spaces within an
; item, default is no files
CopyBeforeFiles = *.ini
```

You can specify either an absolute path or a path relative to the project directory for `CopyBeforeFrom`. If the path contains spaces, you must enclose it in quotes.

`CopyBeforeFiles` lists the files to copy from the `CopyBeforeFrom` directory to your project directory. Files are copied after any pre-conversion actions that delete files from the project directory; see §35.4 [Clearing out old files before converting](#) on page 957.

The file specifications you assign to `CopyBeforeFiles` must be separated by spaces, but no spaces are allowed within a file specification. You can use wildcards in file specifications. File specifications can include absolute or relative paths to indicate where files should be copied from; the default is from the `CopyBeforeFrom` directory, and relative paths are relative to the `CopyBeforeFrom` directory. The destination is always the project directory.

See also:

§35.6.6 [Listing extracurricular files to put in the wrap directory](#) on page 964

35.6 Assembling files for distribution

Mif2Go can create a “wrap” directory for assembling files, copy selected files to the wrap directory for distribution, and optionally clear out the wrap directory first.

In this section:

§35.6.1 [Specifying a wrap directory](#) on page 961

§35.6.2 [Emptying the wrap directory before copying](#) on page 962

§35.6.3 [Listing files to copy to the wrap directory](#) on page 962

§35.6.4 [Understanding when to use other file copy settings](#) on page 963

§35.6.5 [Understanding which files are copied from where](#) on page 963

§35.6.6 [Listing extracurricular files to put in the wrap directory](#) on page 964

35.6.1 Specifying a wrap directory

To specify a wrap directory where **Mif2Go** can place files for distribution:

```
[Automation]
WrapAndShip = Yes
; WrapPath = path to dir to contain the files for distribution,
;   relative OK
WrapPath = path\to\wrap\directory
```

WrapPath can be an absolute path, or a path relative to the project directory. If the directory specified by WrapPath does not exist, **Mif2Go** creates it.

When you first set up a conversion project, by default **Mif2Go** includes the following setting for WrapPath in your new configuration file (see §35.3 [Understanding path values for deliverables](#) on page 957):

```
WrapPath = .\_wrap
```

This path is relative to the project directory. You can change this setting to specify a different path, either relative or absolute. If the path contains spaces, you must enclose it in quotes. If the directory named by WrapPath does not exist, **Mif2Go** creates the directory.

To get rid of WrapPath entirely, you would have to set WrapPath to blank; if there is no setting at all the default value is ._wrap, relative to the project directory.

Note: WrapPath takes effect not only when WrapAndShip=Yes, but also when one of the following is true for the output type specified:

HTML Help: [Automation]CompileHelp=Yes

WinHelp: [Automation]CompileHelp=Yes

JavaHelp: [JavaHelpOptions]FTSCommand=path/to/indexer

Oracle Help: [OracleHelpOptions]FTSCommand=path/to/indexer

To make the wrap directory the same as your project directory, set WrapPath to blank:

```
[Automation]
WrapAndShip=Yes
WrapPath =
```

You might want to use this setting for HTML Help or for WinHelp; see §35.10 [Gathering and processing Help-system files](#) on page 971.

35.6.2 Emptying the wrap directory before copying

To clear out the wrap directory before **Mif2Go** copies files:

```
[Automation]
WrapAndShip=Yes
; EmptyWrapPath = Yes (default, remove all files before copying)
; or No (leave old files in place in WrapPath directory)
EmptyWrapPath=Yes
```

When `EmptyWrapPath=Yes`, provided `WrapPath` does *not* point to the project directory, **Mif2Go** deletes the entire contents of the `WrapPath` directory before copying files.

However, if either of the following is true, **Mif2Go** does not delete anything, regardless of the value of `EmptyWrapPath`:

- neither the configuration file nor any referenced template has a setting for `WrapPath`
- `WrapPath` points to the project directory.

For HTML output types, if `WrapPath` points to the same directory as

`[Graphics]GraphPath`, **Mif2Go** does not delete files unless both `EmptyWrapPath` and `EmptyGraphPath` are set to `Yes`; see §35.7 [Placing graphics files for distribution](#) on page 965.

When `EmptyWrapPath=No`, **Mif2Go** leaves the prior contents of the `WrapPath` directory in place. Orphaned and obsolete files from previous conversion runs could accumulate and find their way into current deliverables. For this reason, it is better to designate a directory for `WrapPath` that is different from the project directory, and set `EmptyWrapPath=Yes`; that way nothing important is lost, and nothing unwanted is delivered.

Note: `EmptyWrapPath` takes effect not only when `WrapAndShip=Yes`, but also when one of the following is true for the output type specified:

HTML Help: `[Automation]CompileHelp=Yes`

WinHelp: `[Automation]CompileHelp=Yes`

JavaHelp: `[JavaHelpOptions]FTSCommand=path/to/indexer`

Oracle Help: `[OracleHelpOptions]FTSCommand=path/to/indexer`

See §35.10 [Gathering and processing Help-system files](#) on page 971.

35.6.3 Listing files to copy to the wrap directory

To list files for **Mif2Go** to copy to the wrap directory (for example):

```
[Automation]
WrapAndShip=Yes
; WrapCopyFiles = list of files to copy, separated by spaces
WrapCopyFiles = *.htm *.js
```

The file specifications you assign to `WrapCopyFiles` must be separated by spaces, but no spaces are allowed within a file specification. You can use wildcards in file specifications. File specifications can include absolute or relative paths to indicate where files should be copied from; the default is from the project directory, and relative paths are relative to the project directory. The destination is always the `WrapPath` directory; see §35.6.1

[Specifying a wrap directory](#) on page 961.

Note: `WrapCopyFiles` takes effect not only when `WrapAndShip=Yes`, but also when one of the following is true for the output type specified:

HTML Help: `[Automation]CompileHelp=Yes`

WinHelp: `[Automation]CompileHelp=Yes`

JavaHelp: `[JavaHelpOptions]FTSCommand=path/to/indexer`

Oracle Help: [OracleHelpOptions]FTSCommand=path/to/indexer

See §35.10 [Gathering and processing Help-system files](#) on page 971.

If no setting for WrapCopyFiles is present, certain files are copied by default from the project directory to the wrap directory. [Table 35-1](#) lists the files that are copied by default for each output type.

Table 35-1 Default files copied from project directory to wrap directory

Output type	Files copied by default from project to WrapPath directory
HTML, XHTML, XML	*.htm *.html *.xhtm *.xhtml *.xml *.js *.dtd *.mod *.ent *.xsd
DITA	*.dita *.ditamap *.bookmap *.dtd *.mod *.ent *.xsd
Eclipse Help	*.htm *.js *.xml
HTML Help	*.htm *.js *.hh? *.h
OmniHelp	*.htm *.js
WinHelp	*.rtf *.hpj *.cnt *.h
Word	*.rtf

Note: Never *move* output files; the originals must remain in the project directory to permit links to work from other files (whenever you convert less than a full document) and from other projects (always).

35.6.4 Understanding when to use other file copy settings

Use settings *other than* WrapCopyFiles for the following:

- JavaHelp and Oracle Help files; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.
- Graphics files and CSS files; see §35.7 [Placing graphics files for distribution](#) on page 965 and §35.8 [Placing CSS or XSL files for assembly](#) on page 969.

35.6.5 Understanding which files are copied from where

If WrapPath has a value *other than the project directory* when you run a conversion, by default **Mif2Go** does the following:

- If the directory designated by WrapPath already exists, and EmptyWrapPath=Yes (the default), **Mif2Go** deletes the prior contents; otherwise **Mif2Go** creates the directory.
- After converting your document, **Mif2Go** copies necessary files from the project directory to the WrapPath directory (and to subdirectories, if appropriate). [Table 35-2](#) lists the files that are copied by default.

For example, to have **Mif2Go** copy to the WrapPath directory only HTML files and just one particular JavaScript file from the project directory, and all other JavaScript files from another directory:

```
[Automation]
WrapAndShip=Yes
WrapPath=.\Done
WrapCopyFiles = *.htm justone.js ..\jsfiles\*.js
```

With these settings, **Mif2Go** also copies graphics files and CSS files from the project directory, unless you specify otherwise; see §35.7 [Placing graphics files for distribution](#) on page 965 and §35.8 [Placing CSS or XSL files for assembly](#) on page 969.

Table 35-2 Files copied by default to the wrap directory

Output type	Files copied by default to the wrap directory, via:		
	WrapCopyFiles	GraphCopyFiles	CssCopyFiles
HTML, XHTML, XML	*.htm *.html *.xml *.dtd *.mod *.ent *.txt *.xsd *.js	*.gif *.jpg *.png	*.css *.xsl
DITA	*.dita *.ditamap *.bookmap *.dtd *.mod *.ent *.xsd	*.gif *.jpg *.png	*.css *.xsl
HTML Help	*.htm *.hh? *.h *.js	*.gif *.jpg *.png	*.css *.xsl
Eclipse Help	*.htm *.js *.xml	*.gif *.jpg *.png	*.css *.xsl
JavaHelp, Oracle Help *	*.xml *.hs *.jhm *.htm *.js	*.gif *.jpg *.png	*.css *.xsl
OmniHelp **	*.htm *.oh?	*.gif *.jpg *.png	*.css *.xsl
WinHelp	*.rtf *.hpj *.cnt *.h	*.bmp *.wmf	<i>Not applicable</i>
Word	*.rtf	*.bmp *.wmf	<i>Not applicable</i>

* Second group of files is copied to the HTML subdirectory. See §11.3.7 [Creating a directory structure for JavaHelp / Oracle Help](#) on page 378.

** For OmniHelp, additional files are copied from a viewer directory; see §10.13 [Assembling OmniHelp files for viewing](#) on page 369.

For all output types, files specified by GraphCopyFiles (default *.gif, *.jpg, *.png, and *.svg for HTML, or *.bmp and *.wmf for RTF) are copied from the project directory to the GraphPath directory; or to a subdirectory, for JavaHelp and Oracle Help; see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.

For all HTML output types, files specified by CssCopyFiles (default *.css and *.xsl) are copied from the project directory to the CssPath directory; see §35.8 [Placing CSS or XSL files for assembly](#) on page 969.

35.6.6 Listing extracurricular files to put in the wrap directory

If your distribution should include other files in addition to those produced by **Mif2Go**, you can have those files copied into the wrap directory after conversion.

```
[Automation]
; CopyAfterFrom = path to directory containing files to add to the
; wrap directory, after moving other files there; for example:
CopyAfterFrom = ..\..\keepers
; CopyAfterFiles = list of files to copy from CopyAfterFrom
; to the wrap directory, default is no files; for example:
CopyAfterFiles = *.bookmap
```

For CopyAfterFrom you can specify either an absolute path or a path relative to the project directory. If the path contains spaces, you must enclose it in quotes.

CopyAfterFiles lists the files to copy from the CopyAfterFrom directory to the wrap directory, after all other files have been placed in the wrap directory.

The file specifications you assign to CopyAfterFiles must be separated by spaces, but no spaces are allowed within a file specification. You can use wildcards in file specifications. File specifications can include absolute or relative paths to indicate where files should be copied from; the default is from the CopyAfterFrom directory, and relative paths are relative to the CopyAfterFrom directory. The destination is always the wrap directory.

35.7 Placing graphics files for distribution

If external graphics files referenced by your document are not already in the project directory, you can have **Mif2Go** copy them there, or to a subdirectory, or to a wrap directory for distribution. This is primarily an issue for HTML output; it is not usually necessary for compiled WinHelp or for normal Word output. For some HTML output types, graphics placement is restricted; see §23.3 [Locating graphics files for HTML](#) on page 704.

In this section:

§35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965

§35.7.2 [Selecting graphics to copy from arbitrary locations](#) on page 966

§35.7.3 [Deleting prior contents of the graphics destination directory](#) on page 967

§35.7.4 [Synchronizing graphics settings for HTML output](#) on page 968

§35.7.5 [Synchronizing graphics settings for RTF output](#) on page 969

See also:

§35.6 [Assembling files for distribution](#) on page 961.

35.7.1 Copying referenced graphics to a distribution directory

When you specify `WrapAndShip=Yes` or designate a `WrapPath` directory, for HTML output **Mif2Go** automatically copies graphics files from the project directory to the directory designated by `[Graphics]GraphPath`; see §23.3 [Locating graphics files for HTML](#) on page 704. **Mif2Go** can also copy graphics files from other locations.

To have **Mif2Go** copy the graphics files referenced by your document to a location relative to the HTML files generated for distribution:

```
[Automation]
; CopyOriginalGraphics = No (default) or Yes (copy graphics to the
; location specified by GraphPath)
CopyOriginalGraphics = Yes
```

When `CopyOriginalGraphics=Yes`, **Mif2Go** copies graphics from wherever they are referenced by your FrameMaker document to one of the following destinations:

- for JavaHelp and Oracle Help for Java, the directory designated by `[JavaHelpOptions]GraphSubdir` (see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379)
- for other HTML output types (and DCL output), the directory designated by `[Graphics]GraphPath`, if any (see §35.7.4 [Synchronizing graphics settings for HTML output](#) on page 968), otherwise the `WrapPath` directory
- for RTF output types (and MIF output), the directory designated by `WrapPath`.

In other words, **Mif2Go** gathers referenced graphics by following the relevant links in the source document; then, using the values of `WrapPath` and `GraphPath`, places those graphics where links in the ready-for-distribution topic files expect to find them.

If your FrameMaker document references graphics files that are in a format not suitable for HTML output, and if you have provided alternates in the same directory with the same file names but a different file extension, you can specify the extension to use for HTML output. See §31.3.1.2 [Substituting graphics files for HTML](#) on page 888.

Note: If you specify `[Automation]OnlyAuto=Yes` (see §35.13 [Postprocessing separately from converting](#) on page 976), and you are relying on

CopyOriginalGraphics to get your graphics files into the wrap directory, they will not arrive; graphics files are *not copied* when OnlyAuto=Yes.

35.7.2 Selecting graphics to copy from arbitrary locations

In addition to (or instead of) having **Mif2Go** gather up copies of the graphics files referenced by your FrameMaker document (see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965), you can have **Mif2Go** copy all or selected graphics files from other locations.

The paths in your FrameMaker document point to the images used during authoring. But different output types require different image formats, so if an image is in the right format for HTML, it is wrong for RTF. You can choose, on a per-project basis, which set of images you want by selecting where to copy them from.

To specify which graphics files to copy and from where:

```
[Automation]
; CopyGraphicsFrom = path to dir containing graphics files,
;   relative OK
CopyGraphicsFrom = path\to\graphics\files
; GraphCopyFiles = list of files to copy from CopyGraphicsFrom,
;   from project directory, and from arbitrary locations.
GraphCopyFiles = *.gif *.jpg G:\special\images\logo.png
```

CopyGraphicsFrom and GraphCopyFiles take effect when WrapAndShip=Yes, CompileHelp=Yes, or FTSCCommand=path\to\indexer (see §35.10 [Gathering and processing Help-system files](#) on page 971).

*Where to get
graphics files*

When you specify a value for CopyGraphicsFrom, graphics files are copied first from the project directory (unless it is the same as the destination directory), then from the directory designated by CopyGraphicsFrom, to one of the destinations listed in §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965. If you specify a relative path for CopyGraphicsFrom, that path is relative to the project directory.

*Where to put
graphics files*

The CopyGraphicsFrom command happens just before the CopyOriginalGraphics command (see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965), and copies to the same place. When WrapAndShip=Yes, that place is the concatenation of the wrap directory (if any) and the value of [Graphics]GraphPath. If no value is specified for GraphPath, files are copied to the wrap directory; if no value is specified for WrapPath, files are copied to a concatenation of the project directory and GraphPath; if neither is specified, files are copied to the project directory.

*Which graphics
files to copy*

You can use GraphCopyFiles to list files to be copied. [Table 35-3](#) shows which graphics files are copied by default for each output type. Files without paths assigned to GraphCopyFiles are always copied first from the project directory, then from the CopyGraphicsFrom directory (if any). If GraphCopyFiles is not specified, or is set to nothing, *all* relevant graphics files are copied from the project directory and then from the CopyGraphicsFrom directory (if any).

Table 35-3 Default graphics files copied for assembly

Output type	Files copied by default from project directory
DCL, MIF	*.bmp *.wmf *.gif *.jpg *.png *.svg *.tif
HTML, XML types	*.gif *.jpg *.png *.svg
RTF types	*.bmp *.wmf

The file specifications you assign to `GraphCopyFiles` must be separated by spaces, and no spaces are allowed within a file specification. You can use wildcards in file specifications, and include absolute or relative paths to indicate where graphics files should be copied from. If you do not specify a path, the default is first from the project directory, then from the `CopyGraphicsFrom` directory (if any). If you specify a relative path, the path is relative to the project directory.

For example, to have **Mif2Go** copy graphics files for standard HTML output from directory `MyGraphics`, parallel to the project directory, to directory `Images`, a subdirectory of the `WrapPath` directory:

```
[Automation]
WrapAndShip=Yes
; WrapPath is relative to the project directory:
WrapPath=.\Final
; CopyGraphicsFrom is relative to the project directory:
CopyGraphicsFrom=..\MyGraphics

[Graphics]
; GraphPath is relative to the WrapPath directory:
GraphPath=./images
```

If you use backslashes for `GraphPath`, **Mif2Go** changes them to forward slashes before inserting references in HTML output, from HTML files to image files. See §23.3 [Locating graphics files for HTML](#) on page 704.

Synchronize with other settings

If you plan to use `CopyGraphicsFrom`, make sure other graphics settings in the configuration file are consistent with the setting for `WrapPath`. See:

§35.7.4 [Synchronizing graphics settings for HTML output](#) on page 968

§35.7.5 [Synchronizing graphics settings for RTF output](#) on page 969

Use system commands instead

As an alternative, you can collect graphics from multiple locations with a series of system commands in a **Mif2Go** macro. For example:

```
[Automation]
SystemWrapCommand=<$GetGraphics>

[GetGraphics]
cd <$$$_currpath>\\wrap
copy "c:\\my graphics\\*.jpg"
copy "c:\\more graphics\\*.jpg"
```

Notice the doubled backslashes (required in **Mif2Go** macros, where backslash is used as an escape character), and the quotes around paths that includes spaces; see §34.4.6 [Supplying system commands in a macro](#) on page 940.

35.7.3 Deleting prior contents of the graphics destination directory

To empty the destination directory before copying graphics files for HTML:

```
[Automation]
WrapAndShip=Yes
; EmptyGraphPath = No (default, leave graphics files in place)
; or Yes (empty GraphPath directory before copying) HTML only
EmptyGraphPath=Yes
```

Note: For JavaHelp and Oracle Help, alternate settings apply; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

`EmptyGraphPath` takes effect when `WrapAndShip=Yes`, `CompileHelp=Yes`, or `FTSCommand=path\to\indexer` (see §35.10 [Gathering and processing Help-system files](#) on page 971).

When EmptyGraphPath=Yes, provided [Graphics]GraphPath does *not* point to the project directory, **Mif2Go** deletes the entire contents of the GraphPath directory before copying files into it. However, if either of the following is true, **Mif2Go** does not delete anything, regardless of the value of EmptyGraphPath:

- No setting is present for GraphPath
- GraphPath points to the project directory.

If WrapPath points to the same directory as GraphPath, **Mif2Go** does not delete files unless both EmptyWrapPath and EmptyGraphPath are set to Yes; see §35.6 [Assembling files for distribution](#) on page 961.

When EmptyGraphPath=No (the default), **Mif2Go** leaves the prior contents of the GraphPath directory in place.

35.7.4 Synchronizing graphics settings for HTML output

For HTML output types, check configuration settings for the following options; their values must reflect the destination, not the origin, of graphics to be copied for distribution:

```
[Graphics]
; StripGraphPath = No (default)
; or Yes (remove path from graphics references)
; GraphPath = path to use (replacing any previous) for all graphics
; GraphPathOverrides = No (default) or Yes (overrides any path
; in Config markers and in [GraphFiles], adding GraphPath
; GraphSuffix = file extension to use for replacement graphics

[GraphFiles]
; Original name (with or without extension) = new name (with
; extension); new name overrides any [Graphics]GraphPath specified

[GraphSuffix]
; old suffix = new suffix, overrides [Graphics]GraphSuffix
```

Note: For JavaHelp and Oracle Help, alternate settings apply; see §11.3.7.2 [Letting Mif2Go set up the directory structure and copy files](#) on page 379.

The value for GraphPath, if present, is ordinarily a path relative to the wrap directory where the generated HTML files are located. This value is inserted in tags in your HTML output, as references to graphics files on the server; see §23.3 [Locating graphics files for HTML](#) on page 704. If your configuration file does not include a setting for GraphPath, by default tags do not include a path, unless you specify a path in [GraphFiles] (see §31.3.1.2 [Substituting graphics files for HTML](#) on page 888).

If WrapPath points to the project directory:

- Set StripGraphPath=Yes
- Remove or comment out any setting for GraphPath
- Set GraphPathOverrides=No (see §31.3.1.3 [Overriding path specifications for referenced graphics](#) on page 888)
- Make sure [GraphFiles] entries and configuration markers *do not* include paths.

If WrapPath points to any directory *except* the project directory:

- Set StripGraphPath=No
- Set GraphPath to point to the WrapPath directory, or to a directory relative to the WrapPath directory
- Set GraphPathOverrides=Yes

See also:

§23.3 [Locating graphics files for HTML](#) on page 704

§23.4.1 [Using referenced graphics without converting](#) on page 706

§31.3 [Replacing and relocating graphics files](#) on page 887

§33.2.9.4 [Overriding graphic properties for HTML](#) on page 929

35.7.5 Synchronizing graphics settings for RTF output

For RTF output types, check configuration settings for the following options; their values must reflect the destination, not the origin, of graphics to be copied to the WrapPath directory:

```
[Graphics]
; FileNames = Retain (default) or Map (in the GraphFiles section)
; FilePaths (for graphics) = Retain (default) or None (strip off)

[GraphFiles]
; types to map, replace extension, old=new for referenced graphics
; specific filenames to replace, old = new, overrides type setting
```

If WrapPath points to the project directory:

- Set FileNames=Map
- Set FilePaths=None
- Make sure [GraphFiles] entries *do not* include paths.

If WrapPath points to any directory other than the project directory:

- Set FileNames=Map
- Set FilePaths=Retain
- Make sure any paths in [GraphFiles] entries point to the WrapPath directory.

See §Table 31-1 [RTF replacement graphics file mappings and locations](#) on page 892.

See also:

§31.3 [Replacing and relocating graphics files](#) on page 887.

35.8 Placing CSS or XSL files for assembly

For HTML output types, when you specify WrapAndShip=Yes and designate a WrapPath directory, **Mif2Go** automatically copies CSS or XSL files from the project directory to the directory designated by [CSS]CssPath. **Mif2Go** can also automatically copy CSS or XSL files from another directory you specify.

To have **Mif2Go** copy CSS or XSL files:

```
[Automation]
WrapAndShip=Yes
; CopyCssFrom = path to directory containing the .css files,
; relative OK
CopyCssFrom=..\css
; CssCopyFiles = list of files to copy from CopyCssFrom and output
; directories to the [CSS]CssPath (which defaults to the WrapPath)
CssCopyFiles=*.css *.xsl
```

CopyCssFrom and CssCopyFiles take effect when WrapAndShip=Yes, CompileHelp=Yes, or FTSCCommand=path\to\indexer (see §35.10 [Gathering and processing Help-system files](#) on page 971).

Say where to get
CSS files

When you specify a value for CopyCssFrom, *.css and *.xsl files are copied first from the project directory (unless it is the same as the destination directory), then from the directory designated by CopyCssFrom, to one of the following destinations:

- the directory designated by [CSS]CssPath, if any (see §22.4.3 [Designating and locating a CSS file](#) on page 686); otherwise,
- the directory designated by WrapPath (see §35.6 [Assembling files for distribution](#) on page 961) or, for JavaHelp and Oracle Help, the .\html subdirectory.

If you specify a relative path for CopyCssFrom, that path is relative to the project directory.

*List CSS files to
be copied*

You can use CssCopyFiles to list CSS or XSL files to be copied; the default files are *.css and *.xsl. Files assigned to CssCopyFiles are always copied first from the project directory to the directory designated by [CSS]CssPath, then from the CopyCssFrom directory (if any). If CssCopyFiles is not present, or is set to nothing, all *.css and *.xsl files are copied from the project directory and then from the CopyCssFrom directory (if any).

The file specifications you assign to CssCopyFiles must be separated by spaces, and no spaces are allowed within a file specification. You can use wildcards in file specifications, and include absolute or relative paths to indicate where files should be copied from. If you do not specify a path, the default is first from the project directory, then from the CopyCssFrom directory. If you specify a relative path, the path is relative to the project directory.

For example, to have **Mif2Go** copy CSS files from directory MyCSS, parallel to the project directory, to directory Styles, a subdirectory of the WrapPath directory:

```
[Automation]
WrapAndShip=Yes
; WrapPath is relative to the project directory:
WrapPath=.\Final
; CopyCssFrom is relative to the project directory:
CopyCssFrom=..\MyCSS

[CSS]
; CssPath is relative to the WrapPath directory:
CssPath=.\Styles
```

35.9 Gathering files for an HTML project: an example

Suppose your file structure looks like this:

D:\AllDocs\CSS	<i>CSS files for all HTML projects</i>
D:\MyDoc	<i>FrameMaker files, projects file, FileID file</i>
D:\MyDoc\Graphics	<i>Graphics</i>
D:\MyDoc\HTML	<i>Mif2Go output files and project configuration file</i>

And you want the files for your HTML project assembled as follows:

D:\MyDoc\HTML_wrap	<i>HTML files should be copied here</i>
D:\MyDoc\HTML_wrap\images	<i>Graphics files should be copied here</i>
D:\MyDoc\HTML_wrap\styles	<i>CSS files should be copied here</i>

Your projects file (.prj, in D:\MyDoc with your FrameMaker files) would specify D:\MyDoc\HTML as the path for **Mif2Go** to use for output. D:\MyDoc\HTML is also where your project configuration file is located.

To get all the files where you want them, in the configuration file you would specify the following:

<u>Section</u>	<u>Setting</u>
[Automation]	<p>WrapPath=. _wrap A location relative to the project directory. You could just as well use the absolute path: WrapPath=D:\MyDoc\HTML_wrap. Notice the <i>backslashes</i> here, which are required for Windows.</p> <p>CopyCssFrom=D:\AllDocs\CSS Where to find the CSS files for this project. Path separators are <i>backslashes</i>.</p> <p>CopyGraphicsFrom=D:\MyDoc\Graphics Where to find graphics for this project. Path separators are <i>backslashes</i>.</p> <p>GraphCopyFiles=*.jpg *.gif Files you want from the CopyGraphicsFrom directory.</p>
[CSS]	<p>CssPath=. \styles Where CSS files should be relative to the HTML files that use them (that is, relative to the WrapPath directory). Mif2Go converts backslashes to forward slashes before writing these references in the HTML files.</p>
[Graphics]	<p>GraphPath=. \images Where the graphics should be relative to the HTML files that reference them (that is, relative to the WrapPath directory). Mif2Go converts backslashes to forward slashes before writing these references in the HTML files.</p>

35.10 Gathering and processing Help-system files

Most Help systems require additional steps after **Mif2Go** generates output files from your FrameMaker document, and before archiving files for distribution. You can have **Mif2Go** automatically do the following:

WinHelp	Run the WinHelp compiler; see §8.2.13 Compiling a WinHelp project on page 250.
HTML Help	Run the HTML Help compiler; see §9.14 Compiling and testing HTML Help on page 333.
OmniHelp	Copy viewer files to the WrapPath directory (needed only if they are not already in the project directory); see §10.13 Assembling OmniHelp files for viewing on page 369.
JavaHelp	Run the full-text-search indexing program; see §11.5.2 Creating a search index for JavaHelp on page 388.
Oracle Help for Java	Run the full-text-search indexing program (although you might not get a usable search index); see §11.5.3 Creating a search index for Oracle Help on page 389.
Eclipse Help	Archive topic files into doc.zip; see §12.8 Packaging Eclipse Help files on page 419.

*Compile WinHelp
or HTML Help*

To direct **Mif2Go** to compile WinHelp or HTML Help:

```
[Automation]
; CompileHelp = No (default, run compiler separately),
; or Yes copy all needed files to the WrapPath, if given,
; then compile with hhc (HTML Help) or hcw (WinHelp).
CompileHelp = Yes
```

By default, CompileHelp=No. See:

§8.2.8 [Setting basic WinHelp options in the configuration file](#) on page 248

§9.14 [Compiling and testing HTML Help](#) on page 333

Index JavaHelp or Oracle Help To direct **Mif2Go** to run the JavaHelp or Oracle Help indexer to create a search index:

```
[JavaHelpOptions] or [OracleHelpOptions]
FTSCommand = path/to/indexer
```

If FTSCommand is missing or is set to blank, **Mif2Go** does not run the indexer. See:

§11.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 387

Certain automation settings are activated

When CompileHelp=Yes or FTSCommand=path/to/indexer, **Mif2Go** acts on those [Automation] settings that need to be processed prior to compilation or indexing, regardless of the setting for WrapAndShip. Then **Mif2Go** runs the appropriate compiler or indexer. [Table 35-4](#) shows which settings are activated.

Table 35-4 Automation settings activated by CompileHelp or FTSCommand

[Automation] setting	Action	Ref.
CopyCssFrom	Copy CSS files from the designated directory	35.8
CopyGraphicsFrom	Copy graphics files from the designated directory	35.7.1
CssCopyFiles	Select only specified CSS files for copying	35.8
DeleteExistingMIF	Delete prior MIF files from the project directory before conversion	35.4.4
EmptyGraphPath	Delete prior copied graphics files before copying	35.7.3
EmptyOutputDir	Delete files from the project directory before conversion	35.4.1
EmptyOutputFiles	Select only specified files to delete from the project directory	35.4.2
EmptyWrapPath	Delete all files from the WrapPath directory before copying	35.6
GraphCopyFiles	Select only specified graphics files for copying	35.7.1
KeepCompileWindow	Keep the compiler window open after compiling: for WinHelp for HTML Help	8.2.13 9.14.1
WrapCopyFiles	Copy only specified files from the project directory	35.6
WrapPath	Directory to which files are copied for compiling and assembling for distribution	35.6
ShipPath	Directory to which compiled or archived files are copied or moved	35.12

Assemble files without compiling or indexing

When CompileHelp=No (the default for WinHelp and HTML Help), or FTSCommand is not specified for JavaHelp or Oracle Help, you must run the compiler or indexer separately. If WrapAndShip=Yes, uncompiled or unindexed Help-system files are assembled for distribution; see §35.2 [Activating and logging production of deliverables](#) on page 956. You might use this combination for WinHelp if you are sending files to be branded by a subcontractor, or to be integrated with other WinHelp systems. For HTML Help, you might send uncompiled files for use on a server.

Assembling and archiving are optional

For WinHelp or HTML Help, you can set the value of WrapPath for compiled Help output to blank (or explicitly to the project directory), because the Help compilers rely on a list of files to include in the compilation. Eliminating a separate wrap subdirectory avoids creating a duplicate set of output files. Also, archiving is not always necessary for compiled Help, because compilation itself creates a compressed deliverable.

Convert first, compile and archive later

Having **Mif2Go** run either Help compiler as part of the conversion can be problematic for large projects. Instead, you can do the conversion as a first step, then run **Mif2Go** again to compile and prepare the deliverable; see §35.13 [Postprocessing separately from converting](#) on page 976.

35.11 Archiving deliverables

To archive output files assembled for distribution, **Mif2Go** can automatically run a command-line archiving program such as `pkzip.exe`, or WinZip command-line add-on `wzip.exe`. **Mif2Go** composes and executes an archiving command based on values you supply for the archiving program and its parameters. For example, for `wzip.exe` the command and basic parameters are as follows:

```
wzip [options] zipfile [files...]
```

Mif2Go uses the following settings to put together the components of this command:

<u>Component</u>	<u>Mif2Go archive setting(s)</u>	<u>Reference</u>
wzip	ArchiveCommand	35.11.1
[options]	ArchiveStartParams	35.11.2
zipfile	ArchiveName, ArchiveVer, ArchiveExt	35.11.3
[files...]	ArchiveEndParams	35.11.2

In this section:

- [§35.11.1 Specifying an archiving command](#) on page 973
- [§35.11.2 Supplying parameters for the archiving command](#) on page 973
- [§35.11.3 Specifying archive file name and optional version](#) on page 974

35.11.1 Specifying an archiving command

To have **Mif2Go** archive files assembled for distribution:

```
[Automation]
WrapAndShip=Yes (or CompileHelp=Yes)
; ArchiveCommand = zip command, without parameters
ArchiveCommand=pkzip
```

ArchiveCommand must include the absolute path to the location of the archiving program on your system, unless that location is already on the system PATH. If the path contains spaces, you must enclose the path (including the command name) in quotes. For example, the archiving command setting for the **Mif2Go User's Guide** is:

```
ArchiveCommand = "g:\program files\winzip\wzip"
```

ArchiveCommand has no default value; if you do not specify a value, **Mif2Go** does no archiving, and the remaining Archive* settings are moot.

ArchiveCommand takes effect only when at least one of the following is true:

- WrapAndShip=Yes (see [§35.2 Activating and logging production of deliverables](#) on page 956)
- OnlyAuto=Yes (see [§35.13 Postprocessing separately from converting](#) on page 976).

If WrapPath is set to blank, the archiving command works on whatever is in the project directory; when this is the case, unless you include exactly the right parameters, you might get a mess. See [§35.11.2 Supplying parameters for the archiving command](#) on page 973.

35.11.2 Supplying parameters for the archiving command

To provide values for parameters (other than the archive file name) required by the archiving program:

```
[Automation]
; ArchiveStartParams = parameters preceding name of archive file
ArchiveStartParams=-add
```



```
; ArchiveEndParams = parameters following name of archive file
ArchiveEndParams=*. *
```

For parameters that are to be passed to an archiving program, observe the following:

- Do not enclose parameter values in quotes.
- Use backslashes as separators in path-name parameters.
- Use a dash (“-”) instead of a forward slash to prefix a command option.

*Starting
parameters*

ArchiveStartParams specifies any parameters to ArchiveCommand that must *precede* the name of the archive file, such as command option -add for **pkzip** or -a (the default) for **wzip**. For example, the starting-parameter setting for the **Mif2Go User's Guide** is simply:

```
ArchiveStartParams =
```

*Ending
parameters*

ArchiveEndParams specifies any parameters to ArchiveCommand that must *follow* the name of the archive file, such as *. * for **pkzip** or **wzip**. For example, the ending-parameter setting for the Eclipse Help version of the **Mif2Go User's Guide** is:

```
ArchiveEndParams = doc.zip *.xml
```

*Archiving directly
from the project
directory*

If you are archiving from the project directory instead of from a separate directory designated by WrapPath (see §35.6 [Assembling files for distribution](#) on page 961), it is better to enumerate the files (at least by extension) to include in the archive. If you specify ArchiveEndParams=*. *, you might end up with .ref, .ini, .grx, and other unwanted files in the archive. For example, for an HTML project to be archived from the project directory you might specify the following:

```
ArchiveEndParams = *.htm *.css *.gif *.jpg *.png
```

For the HTML Help version of the **Mif2Go User's Guide**, which does not use a WrapPath directory, the setting specifies each file to be included:

```
ArchiveEndParams = ugmif2go.chm
```

35.11.3 Specifying archive file name and optional version

To specify a name for the archive file:

```
[Automation]
; ArchiveName = base name for archive to be created
ArchiveName = MyProj
; ArchiveVer = version number (if any) to be appended to ArchiveName,
; default is the system configuration output-type identifier
ArchiveVer = beta
; ArchiveExt = file extension to be appended, usually zip
ArchiveExt = zip
```

The full name of the archive file is a concatenation of the following:

[Archive file base name](#)

[Archive version](#)

A period (dot)

[Archive file extension.](#)

*Archive file base
name*

ArchiveName is the base file name of the archive to be created. For example, the base name for the archive of the RTF version of the **Mif2Go User's Guide** is:

```
ArchiveName = UGrtf
```

The value you specify for ArchiveName must not contain spaces. The default value of ArchiveName depends on the output type. The default base name of any deliverable (archive or compiled Help system) is the base name of the project. For most Help systems, this is the Help project file name; for other output types, it is the **Mif2Go** project file

name. [Table 35-5](#) shows the source of the default base file name of the archive for each output type.

Table 35-5 Default base file name for deliverables archive

Output type	Source of default base file name for archive	Ref.
HTML Help	[MSHtmlHelpOptions]HHPFileName	9.3.7
JavaHelp, Oracle Help	[JavaHelpOptions]HSFileName	11.3.8
OmniHelp	[OmniHelpOptions]ProjectName (<i>without prefix or suffix</i>)	10.3.3
WinHelp	[HelpOptions]HPJFileName	8.2.8
Eclipse Help	plugin (<i>literally</i>)	12.8.5
All other output types	[Setup]PrjFileName (<i>without path</i>)	C.3

Archive version ArchiveVer is an optional version identifier to be appended to ArchiveName, and may include any alphanumeric characters allowed in file names; see §1.1.2 [File, directory, and path names](#) on page 51. If you do not specify a value for ArchiveVer, **Mif2Go** uses a default output-type identifier as the value; for example, OH for OmniHelp. Output-type identifier values are located in system configuration files for each output type.

Archive file extension ArchiveExt is the file extension for the type of archive to be created (without the leading period); usually zip or jar. The default depends on the value of ArchiveCommand (see §35.11.1 [Specifying an archiving command](#) on page 973). If ArchiveCommand contains **jar**, the default extension is .jar; otherwise the default extension is .zip. **Mif2Go** provides the leading period.

35.12 Placing deliverables in a shipping directory

You can have **Mif2Go** copy or move deliverable files to a separate directory for shipping, or sharing, or storage.

In this section:

§35.12.1 [Specifying a shipping directory for deliverables](#) on page 975

§35.12.2 [Understanding which files are placed in the shipping directory](#) on page 976

§35.12.3 [Choosing whether to copy or move deliverables](#) on page 976

35.12.1 Specifying a shipping directory for deliverables

To have **Mif2Go** place compiled or archived deliverables in a shipping directory:

```
[Automation]
WrapAndShip=Yes
; ShipPath = path to dir to contain final result file of archiving
; or of compilation (.chm, .jar, or .zip), may be the same for
; several projects.
ShipPath=path\to\deliverables
```

ShipPath takes effect only when WrapAndShip=Yes, and only if you have specified a value for ArchiveCommand. See §35.2 [Activating and logging production of deliverables](#) on page 956.

When you first set up a conversion project, **Mif2Go** includes the following setting for ShipPath in your new configuration file (see §35.3 [Understanding path values for deliverables](#) on page 957):

```
ShipPath=..\..\_ship
```

You can change this setting to specify a different path. If the path contains spaces, you must enclose it in quotes. If the directory specified by `ShipPath` does not exist, **Mif2Go** creates this directory for you.

35.12.2 Understanding which files are placed in the shipping directory

When `WrapAndShip=Yes` and you specify a value for `ShipPath`, which files get placed in the `ShipPath` directory depends on the following factors:

- whether or not you also specify a value for `ArchiveCommand`
- the output type of your **Mif2Go** project.

If you specify a value for `ArchiveCommand` (see §35.11 [Archiving deliverables](#) on page 973), **Mif2Go** copies (or moves) any resulting archive to the `ShipPath` directory after all other processing is finished.

If you do not specify a value for `ArchiveCommand`, what gets placed in the `ShipPath` directory depends on the output type. Compiled or JARred Help systems are copied or moved; other output types are not:

<u>Output type</u>	<u>File(s) placed in ShipPath when no ArchiveCommand is specified</u>
HTML Help	<i>MyProj.chm</i>
JavaHelp, Oracle Help	<i>MyProj.jar</i>
WinHelp	<i>MyProj.hlp, MyProj.cnt</i>
All other output types	<i>None</i>

35.12.3 Choosing whether to copy or move deliverables

When `WrapAndShip=Yes` and a value is specified for `ShipPath`, by default **Mif2Go** copies deliverables to the `ShipPath` directory, leaving the originals in the `WrapPath` directory.

To have **Mif2Go** move deliverables instead of copying them:

```
[Automation]
WrapAndShip=Yes
; MoveArchive = No (default, copy archive to ShipPath) or Yes (move
; archive to ShipPath instead of copying it)
MoveArchive=Yes
```

When `MoveArchive=Yes`, deliverables are moved to the `ShipPath` directory and the originals are deleted from the `WrapPath` directory.

When `MoveArchive=No`, deliverables are copied to the `ShipPath` directory, and the originals remain in the `WrapPath` directory.

`MoveArchive` takes effect only when `WrapAndShip=Yes` (see §35.2 [Activating and logging production of deliverables](#) on page 956) and `ShipPath` has a non-blank value.

35.13 Postprocessing separately from converting

If you have already converted a document and the results are still in the project directory, you can have **Mif2Go** carry out postprocessing steps without going through the entire conversion again. These steps can include:

- compiling for WinHelp or HTML Help
- running a search indexer and creating a JAR file for JavaHelp or Oracle Help

- any of the automation options available when you set `WrapAndShip=Yes` (see §35.2 [Activating and logging production of deliverables](#) on page 956) *except* `CopyOriginalGraphics`; see §35.7.1 [Copying referenced graphics to a distribution directory](#) on page 965.

To postprocess conversion results independently of conversion:

```
[Automation]
WrapAndShip=Yes
; OnlyAuto = No (default) or Yes (run only automation commands,
; rather than the full conversion)
OnlyAuto = Yes
```

When `OnlyAuto=Yes`, **Mif2Go** processes options specified in the `[Automation]` section of the configuration file, without first performing any document conversion.

Note: Commands assigned to `SystemStartCommand` are *not* run when `OnlyAuto=Yes`; see §34.4 [Executing operating-system commands](#) on page 937.

`OnlyAuto=Yes` takes effect only when at least one of the following is true:

- `WrapAndShip=Yes`
- `CompileHelp=Yes`
- A value is specified for `FTSCCommand` (for JavaHelp or Oracle Help output) or for `JARCommand` (for JavaHelp).

When `OnlyAuto=No`, **Mif2Go** runs the conversion before processing options specified in the `[Automation]` section.

*Compilation is
included for
WinHelp, HTML
Help*

If the output type is WinHelp or HTML Help and you set `CompileHelp=Yes` (or you check **Compile Help** on the *Export* dialog), **Mif2Go** runs the appropriate compiler before placing the deliverable(s) in a shipping directory; see §35.10 [Gathering and processing Help-system files](#) on page 971.

Note: If you set `CompileHelp=No` when `OnlyAuto=Yes` (because you compiled your Help system in a previous run), be sure to set `EmptyWrapPath=No`; otherwise, your compiled Help system will be swept away before anything else happens.

*Indexing search
terms is included
for JavaHelp,
Oracle Help*

If the output type is JavaHelp or Oracle Help and you specify a value for `FTSCCommand` in the configuration file, **Mif2Go** runs the designated indexer before archiving the deliverables and placing them in a shipping directory; see §11.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 387.

36 Converting via runfm

Omni Systems **runfm** is an asynchronous FDK client for FrameMaker version 6.0 and above. Use **runfm** to run one or a series of unattended **Mif2Go** conversions from outside FrameMaker. This section shows you how.

***Note:** Unattended conversions are meant for production use, where the process has first been worked out interactively. To use runfm, you must have Mif2Go installed, and you may need Windows administrator privileges.*

In this section:

- §36.1 [Designing a project for unattended operation](#) on page 979
- §36.2 [Setting up FrameMaker for unattended operation](#) on page 980
- §36.3 [Understanding runfm command-line syntax](#) on page 980
- §36.4 [Using runfm for Mif2Go conversions](#) on page 982
- §36.5 [Troubleshooting runfm processes](#) on page 987
- §36.6 [Comparing runfm with the DCL command-line filter](#) on page 991
- §36.7 [Operating runfm across a network](#) on page 992
- §36.8 [Using runfm for other FrameMaker plug-ins](#) on page 993

See also:

- §34 [Automating Mif2Go conversions](#) on page 933
- §35 [Producing deliverable results](#) on page 955
- §37 [Converting via DCL](#) on page 995

36.1 Designing a project for unattended operation

You can design a **Mif2Go** project that uses all of the following:

- system commands to check files out of source control before conversion
- post-processing settings to assemble, compile, and archive after conversion
- system commands to check files back into source control
- additional system commands to handle any other post-processing steps.

When you use **runfm** to operate **Mif2Go** from outside FrameMaker, the entire conversion project (or multiple conversion projects), as well as document printing, can be accomplished via unattended operation, including scheduled operation and remote operation. You can invoke **runfm** from a `.bat` file, or from the Windows Control Panel via Scheduled Tasks, so you can use **runfm** to run multiple conversion projects overnight; see §36.5.6 [Running a series of Mif2Go conversions](#) on page 990.

***Note:** You must have Mif2Go installed to use runfm.*

See also:

- §34 [Automating Mif2Go conversions](#) on page 933
- §35 [Producing deliverable results](#) on page 955

36.2 Setting up FrameMaker for unattended operation

Before you can use **runfm**, you must set up FrameMaker for automatic operation. *You must have Windows administrator privileges to do so*, because Windows records in the registry the setting required to find the executable to run.

As Administrator, execute the following command, either in a command window or via **Start > Run...**:

```
"path\to\FramerMaker" -progid:FrameMaker.M2G -auto
```

where:

- `path\to\` is the location where FrameMaker is installed on your system; enclose the entire path and file name in double quotes if the path contains any spaces; you can omit the path if FrameMaker is on your system execution PATH
- `-progid` is a required RPC (Remote Procedure Call) server identifier; for using **runfm**, the default identifier is `FrameMaker.M2G`; the colon between `-progid` and the identifier is required
- `-auto` allows **runfm** to start FrameMaker from the command line; without this option, FrameMaker must already be open for **runfm** to work.

After you set up FrameMaker this way, you can use **runfm** to run, from a command line, any FrameMaker plug-in that is set up to receive RPC notifications.

Despite its name, the `-progid` option has nothing to do with which FrameMaker plug-in you want to run. You can provide any string as the value for `-progid`, as long as you start FrameMaker with that string first, then specify the same value when you execute **runfm**. If you specify `-progid:FrameMaker.M2G` when you set up FrameMaker, you can omit this option entirely when you use **runfm** to run **Mif2Go**.

These settings become effective after you exit FrameMaker. Once set, they remain set in the Windows Registry (in `HKEY_LOCAL_MACHINE/Software/Classes`), until/unless you set up FrameMaker again, perhaps with a different value for `-progid`.

Note: On a 64-bit version of Windows, entries for 32-bit applications such as FrameMaker are buried in a subkey of a subkey: the `Wow6432Node` key located below the primary `Software` key. Expand this key to see the 32-bit keys and values.

See also:

§36.4 [Using runfm for Mif2Go conversions](#) on page 982

§36.8 [Using runfm for other FrameMaker plug-ins](#) on page 993

36.3 Understanding runfm command-line syntax

The syntax for **runfm** is as follows:

```
runfm [ -progid FrameMaker.M2G ]
      [ -remote systemname ]
      [ -book [ path\to\yourbook.book ] ]
      [ -doc [ path\to\yourdoc.fm ] ]
      [ -project "your Mif2Go project name" ]
      [ -client OmniBookExport ]
      [ -print ( book | doc ) [ path\to\printfile.prn ]
      [ -reverse ( yes | no ) ] ]
      [ [ -pdf ( book | doc ) [ path\to\pdf\file.pdf ] ]
      [ [ -pdfsave ( book | doc ) [ path\to\pdf\file.pdf ] ] ]
```

```
[ -printer "name of printer to set via SetPrint" ]
[ -close ( book | doc | all ) ]
[ -diag ]
[ -log mif2go.log ]
```

Type the **runfm** command and options all on one line, at a command prompt. You can use either “/” or “-” as the switch indicator for **runfm** command-line options. If you specify no options at all, or if you specify **-help** (or **/help**) or **-?** (or **/?**), you get a message about usage, then **runfm** exits.

To use the **-print**, **-printer**, and **-pdf** options, you must have the SetPrint plug-in for FrameMaker installed. You can download SetPrint from Sundorne Communications:

<http://www.sundorne.com/FrameMaker/Freeware/setPrint.htm>

Table 36-1 describes **runfm** command-line options and their arguments; see §36.4 Using **runfm** for Mif2Go conversions on page 982 for additional information.

Table 36-1 Command-line options for runfm

Option	Description	Ref.
-progid	Names whatever identifier you used when you set up FrameMaker (see Setting up FrameMaker for unattended operation); the default is <code>FrameMaker.M2G</code> ;	36.4.1.1
-remote	Names a different system (in the same domain), where FrameMaker is to be run; the same user must be logged into, and FrameMaker must already be open on, the specified system.	36.4.1.3
-book	Optionally specifies the full absolute path to a book to be converted, or to a book that contains a chapter specified by the -doc option (otherwise, the book currently active in FrameMaker); quote the path name if it contains any spaces.	36.4.1.2
-doc	Optionally specifies the full absolute path to a document to be converted (otherwise, the document currently active in FrameMaker); quote the path name if it contains any spaces.	36.4.1.2
-project	Names a Mif2Go project (<i>not</i> the name of a <code>.prj</code> file); this name must be <i>listed</i> in the <code>.prj</code> file for the specified book or document; quote the name if it contains any spaces. (For plug-ins other than Mif2Go , specifies the text the plug-in expects; see §36.8 Using runfm for other FrameMaker plug-ins on page 993.)	36.4.2
-client	Specifies the <code>ClientName</code> of a FrameMaker plug-in to run; the default is <code>OmniBookExport</code> ; you do not need this option for Mif2Go .	36.8
-print	Optionally specifies the full absolute path to a file to which the active book (-print book) or document (-print doc) will be printed, using the printer specified by the -printer option or the default FrameMaker printer; quote the path name if it contains any spaces; cannot be used in the same run as -pdf ; <i>requires SetPrint</i> .	36.4.3.1
-reverse	When used with -print , specifies the print order: -reverse yes causes output to be printed last sheet first -reverse no causes output to be printed first sheet first	36.4.3.1
-pdf	Optionally specifies the full absolute path to a PDF file to which the active book (-print book) or document (-print doc) will be moved, after printing via Adobe PDF; quote the path name if it contains any spaces; cannot be used in the same run as -print ; <i>requires SetPrint</i> .	36.4.3.2
-pdfsave	Same as -pdf , except executes Save As PDF (FrameMaker version 8 and later versions only). Do not use with -close all .	36.4.3.3
-printer	Names the printer to use when you specify the -print option; works only if you have installed SetPrint.	36.4.3.4

Table 36-1 Command-line options for runfm

Option	Description	Ref.
-close	Causes FrameMaker (or just the book, or the just document) to close after the conversion: -close book closes the book file -close doc closes the active document -close all closes all open documents and FrameMaker.	36.4.4
-diag	Causes runfm to write more diagnostic messages to the FrameMaker console window; intended primarily for Mif2Go developers.	36.5.2
-log	Names a text file to which the contents of the FrameMaker console window are appended when you also specify a -close option; the default log file name is <code>mif2go.log</code> .	36.5.2

36.4 Using runfm for Mif2Go conversions

To use **runfm**, you must have **Mif2Go** installed on your system. See §1.3.3 [Install Mif2Go](#) on page 56. Also, you may need Windows administrator privileges on the system.

After you set up FrameMaker for unattended operation (see §36.2 [Setting up FrameMaker for unattended operation](#) on page 980), you can use the **runfm** command to start FrameMaker from the command line and run **Mif2Go** automatically. However, before you test this process, *make sure the same operation works when you start it manually from within FrameMaker*. If it does not work that way, **runfm** will not work either.

In this section:

- §36.4.1 [Locating FrameMaker executable and files](#) on page 982
- §36.4.2 [Identifying your Mif2Go project](#) on page 983
- §36.4.3 [Configuring runfm output](#) on page 984
- §36.4.4 [Closing FrameMaker files after conversion](#) on page 987
- §36.5.5 [Running a single Mif2Go conversion or print job](#) on page 989
- §36.5.6 [Running a series of Mif2Go conversions](#) on page 990
- §36.5.7 [Including runfm in a multi-step or scheduled process](#) on page 991

36.4.1 Locating FrameMaker executable and files

In this section:

- §36.4.1.1 [Locating FrameMaker: runfm -progid option](#) on page 982
- §36.4.1.2 [Specifying FrameMaker file paths: runfm -book, -doc options](#) on page 983
- §36.4.1.3 [Using FrameMaker remotely: runfm -remote option](#) on page 983

36.4.1.1 Locating FrameMaker: runfm -progid option

If you are running FrameMaker on the same system as **runfm**, for **-progid** use the same text string you specified for this option when you set up FrameMaker for unattended operation; see §36.2 [Setting up FrameMaker for unattended operation](#) on page 980. However, you do not need the **-progid** option at all to use **runfm** for **Mif2Go** projects, provided you first set up FrameMaker with the default value, `FrameMaker.M2G`.

If you are using FrameMaker across a network, you will need a different value for **-progid**; see §36.7 [Operating runfm across a network](#) on page 992.

36.4.1.2 Specifying FrameMaker file paths: `runfm -book`, `-doc` options

When you use the `-book` and `-doc` options, if you specify a book or a document or both, you *must* use full absolute paths to the book and document files, even if you invoke `runfm` from the same directory where those files are located. If either path contains any spaces, enclose the path in quotes.

When you use the `-book` option or the `-doc` option without specifying *any value at all* for `path/to/yourbook.book` or `path/to/yourdoc.fm`, `runfm` uses whatever book or document is already active in FrameMaker, if any.

If you are updating a file in a book and you specify a project listed in `bookname.prj` (see §36.4.2 [Identifying your Mif2Go project](#) on page 983), use both `-book` and `-doc` options. When you supply values for both `-book` and `-doc`, `runfm` opens both, book first.

If you specify `-doc` but not `-book`, and a book that contains the specified document file is also open in FrameMaker, **Mif2Go** treats the document conversion as an update to the corresponding book project. So if you really want the document to be standalone, make sure you first close any book that contains it.

If you execute a series of `runfm` commands without `-close` on the same book or document, you do not have to repeat the `-book` or `-doc` option after the first command; see §36.5.6 [Running a series of Mif2Go conversions](#) on page 990. As long as `runfm` does not close FrameMaker or open a new book or document, the previous book or document is still active.

See also:

§36.4.2 [Identifying your Mif2Go project](#) on page 983

§36.4.4 [Closing FrameMaker files after conversion](#) on page 987

§36.5.6 [Running a series of Mif2Go conversions](#) on page 990

36.4.1.3 Using FrameMaker remotely: `runfm -remote` option

You need the `-remote` option for **Mif2Go** projects only if you are using FrameMaker across a network, in which case see §36.7 [Operating runfm across a network](#) on page 992.

36.4.2 Identifying your Mif2Go project

When you specify `-project myproject`, the name `myproject` is whatever name you gave your **Mif2Go** project when you set it up via the *Choose Project* dialog (see §3.3 [Creating a Mif2Go conversion project](#) on page 78).

Note: The name `myproject` is *not* the name of your **Mif2Go** project file; instead, `myproject` is a name that is *listed in* your **Mif2Go** project file. The names of all the **Mif2Go** conversion projects you have set up for a given book or document are listed in a file with extension `.prj`, located in the same directory as the FrameMaker files. See §3.3 [Creating a Mif2Go conversion project](#) on page 78 and §C.2.1 [Conversion files created during set-up](#) on page 1020.

The name `myproject` must be present in the `.prj` file for the book (or document) specified on the `runfm` command line (or already open in FrameMaker). If `myproject` contains any spaces, enclose the name in quotes.

If you are updating a document using a book project (a project listed in `bookname.prj`), you must make sure the book is also open; see §36.4.1.2 [Specifying FrameMaker file paths: `runfm -book`, `-doc` options](#) on page 983. When you specify a value for `-doc` (even if you also specify a value for `-book`), `runfm` first looks for `docname.prj`, in whatever

directory you specified in the absolute path to `docname.fm`. If there is no `docname.prj` file in the specified directory, and a book is open in FrameMaker, **runfm** looks for `bookname.prj`. If this search also fails, **runfm** looks for a file named `mif2go.prj`.

The **-project** option is required to run **Mif2Go**. If you omit this option, **runfm** merely opens FrameMaker and any book or document specified, then stops. This can be a useful feature, if you frequently start FrameMaker with the same book and document for editing or testing. You could even create a desktop shortcut to use **runfm** this way.

See also:

§3.3 [Creating a Mif2Go conversion project](#) on page 78

§36.4.1.2 [Specifying FrameMaker file paths: runfm -book, -doc options](#) on page 983

§C.2.1 [Conversion files created during set-up](#) on page 1020

36.4.3 Configuring runfm output

In this section:

§36.4.3.1 [Configuring print output: runfm -print and -reverse options](#) on page 984

§36.4.3.2 [Configuring PDF output: runfm -pdf option](#) on page 985

§36.4.3.3 [Configuring PDF output: runfm -pdfsave option](#) on page 986

§36.4.3.4 [Specifying output via SetPrint: runfm -printer option](#) on page 987

36.4.3.1 Configuring print output: runfm -print and -reverse options

When you use the **-print** option, **runfm** prints the specified book or document (see §36.4.1.2 [Specifying FrameMaker file paths: runfm -book, -doc options](#) on page 983); or if none is specified, prints the book or document that is already active in FrameMaker. If you also use the **-project** option (see §36.4.2 [Identifying your Mif2Go project](#) on page 983), the conversion project runs after printing is finished.

Note: You may not specify both **-print** and **-pdf** in the same invocation of **runfm**.

Also, for printing to work without user intervention, you must make sure the default settings for your printer do not require such action; see §36.4.3.4

[Specifying output via SetPrint: runfm -printer option](#) on page 987.

*Name a print file,
or print directly*

If you name a print file, **runfm** prints to the file instead of to a printer; you *must* specify a full absolute path to the file, enclosed in quotes if the path contains any spaces. The print-file path is set by **runfm** as the FrameMaker print-to-file destination for the book or document, with **Print to File:** checked. However, if the name of the print file ends with `.pdf`, **runfm** assumes that the printer is Adobe PDF, and treats the print file as though you had specified **-pdf** instead of **-print**; see §36.4.3.2 [Configuring PDF output: runfm -pdf option](#) on page 985. If you do not name a print file, **runfm** prints directly to the printer.

*Make sure output
is produced in the
correct order*

When you print to file, to make sure pages come out in the order you expect, also specify **-reverse yes** (for last sheet first) or **-reverse no** (for first sheet first). Certain FrameMaker files seem to have this print option set incorrectly when opened with **runfm**; for example, files originally created in FrameMaker 6 or an earlier version, then converted to FrameMaker 7. (The problem seems not to arise when you print directly to a printer.) After printing to file, **runfm** restores the original value of **Last Sheet First**. Because this might not be the correct value, if you plan to save a file opened by **runfm**, first check how this option is set in the FrameMaker *Print* dialog, and reset it if necessary.

*Make sure the
printer you want
is the current
printer*

Unless you also specify a printer with the **-printer** option *and* have SetPrint installed (see §36.4.3.4 [Specifying output via SetPrint: runfm -printer option](#) on page 987), **runfm** uses whatever printer driver is the current default in FrameMaker. If you do not specify a

printer, or you have not installed SetPrint, before using the **-print** option with **runfm** you must make sure the printer you want to use is either the default Windows printer, or is set as the FrameMaker default printer.

See also:

§36.4.1.2 [Specifying FrameMaker file paths: runfm -book, -doc options](#) on page 983

§36.4.2 [Identifying your Mif2Go project](#) on page 983

§36.4.3.2 [Configuring PDF output: runfm -pdf option](#) on page 985

§36.4.3.4 [Specifying output via SetPrint: runfm -printer option](#) on page 987

36.4.3.2 Configuring PDF output: runfm -pdf option

When you use the **-pdf** option, if the default printer for FrameMaker is Adobe PDF (or if you also specify **-printer** "Adobe PDF"), **runfm** generates a PDF file either from the specified book or document (see §36.4.1.2 [Specifying FrameMaker file paths: runfm -book, -doc options](#) on page 983), or if none is specified, from the book or document that is already active in FrameMaker. If you also use the **-project** option (see §36.4.2 [Identifying your Mif2Go project](#) on page 983), the conversion project runs after PDF generation is finished.

Note: You may not specify both **-print** and **-pdf** in the same invocation of **runfm**.

PDF output is assumed to be in My Documents

If you name a PDF file, you *must* specify a full absolute path, enclosed in quotes if the path contains any spaces. After FrameMaker generates PDF output, **runfm** moves the resulting PDF file from My Documents to the specified location, and gives the file the specified name. *If FrameMaker writes PDF files somewhere other than My Documents, you must move the file yourself, after runfm finishes.*

If you specify a path but no PDF file name, **runfm** names the PDF file *bookname.pdf* or *docname.pdf*, and moves it to the specified location. For example, for a single-file FrameMaker document, this command:

```
runfm -doc E:\Mydoc.fm -printer "Adobe PDF" -pdf doc E:\PDFs
```

would produce *Mydoc.pdf* in My Documents, then move it to *E:\PDFs\Mydoc.pdf*.

Make sure Adobe PDF is the current printer

Unless Adobe PDF is the default Windows printer, or is set as the FrameMaker default printer, you must install SetPrint and specify **-printer** "Adobe PDF" (see §36.4.3.4 [Specifying output via SetPrint: runfm -printer option](#) on page 987). If you do not specify **-printer** "Adobe PDF", or you have not installed SetPrint, before using the **-pdf** option with **runfm** you must make sure Adobe PDF is either the default Windows printer, or is set as the FrameMaker default printer.

Note: **runfm** does *not* use FrameMaker **Save As**.

Configure Adobe PDF for no prompts

For **runfm** to generate PDF without user intervention, you must also make sure that no prompts are required:

1. On the system where FrameMaker is running, go to **Start > Control Panel > Printers**.
2. Right-click the entry for Adobe PDF, and choose **Printing Preferences...**; the *Adobe Printing Preferences* dialog opens.
3. On the **Adobe PDF Settings** tab, *uncheck* the following options:
 - View Adobe PDF Results**
 - Prompt for Adobe PDF filename** (*if this item is present*)
 - Ask to Replace existing PDF file**

If you are using Acrobat 7 or a later version, you might find "Prompt for Adobe PDF filename" as a choice in a drop-down list for **Adobe PDF Output Folder**, instead of as

a checkbox item; if this is the case, choose the following item instead from the drop-down list:

My Documents*.pdf

4. Click **OK** to dismiss the *Adobe Printing Preferences* dialog. Under **Location** in the Printers window, you should now see My Documents listed for Adobe PDF.

*runfm sets
FrameMaker print
options for PDF*

When you specify **-pdf**, **runfm** sets the following print options in FrameMaker:

Thumbnails	No
Skip Blank Pages	No
Last Sheet First	No
Copies:	1
Odd-Numbered Pages	Yes
Even-Numbered Pages	Yes
Scale:	100%
Print Separations	No

However, **runfm** does *not* check **Print to File**, because doing so would not yield a PDF in one step. After a PDF file is created, **runfm** restores the original values of these FrameMaker print options.

*For PostScript,
use -print instead
of -pdf*

If what you really want is PostScript output (for example, if you are using a watched directory to distill PostScript to PDF), use the **-print** option instead of the **-pdf** option, specify **-printer** "Adobe PDF", and give the output file extension .ps; also specify **-reverse** no to make sure output is in the correct order (see §36.4.3.1 [Configuring print output: runfm -print and -reverse options](#) on page 984). For example, with book file UG.book already open in FrameMaker:

```
runfm -print book E:\PS\In\UG.ps -reverse no -printer "Adobe PDF"
```

See also:

- §36.4.1.2 [Specifying FrameMaker file paths: runfm -book, -doc options](#) on page 983
- §36.4.2 [Identifying your Mif2Go project](#) on page 983
- §36.4.3.1 [Configuring print output: runfm -print and -reverse options](#) on page 984
- §36.4.3.3 [Configuring PDF output: runfm -pdfsave option](#) on page 986
- §36.4.3.4 [Specifying output via SetPrint: runfm -printer option](#) on page 987

36.4.3.3 Configuring PDF output: runfm -pdfsave option

If you are using FrameMaker version 8 (fully patched) or a later version, you can direct **runfm** to generate PDF output via the FrameMaker **Save As PDF** function instead of the **Print** function. You can avoid worrying about whether Adobe PDF is the default printer, and you can continue to use Microtype TimeSavers, which works with Distiller X only if Distiller is run with the **-f** option. **Save As PDF** from within FrameMaker versions 8 (patched), 9, 10, and 11 all use Distiller with the **-f** option.

As with **-pdf**, do not specify **-print** in the same command.

If you specify **-close** all in the same command as **-pdfsave**, FrameMaker leaves debris behind (a .tps file and a .tpdf file). However, you can use **-close** book without this problem.

See also:

- §36.4.3.2 [Configuring PDF output: runfm -pdf option](#) on page 985

36.4.3.4 Specifying output via SetPrint: runfm -printer option

When you use the **-printer** option, if you also specify **-print** or **-pdf**, **runfm** calls the Sundorne Communications SetPrint plug-in for FrameMaker to use the specified printer. However:

- If you have not installed SetPrint, this option does not work.
- If you have already used SetPrint to designate the printer you want **runfm** to use as the default printer for FrameMaker, you do not need the **-printer** option.

Use SetPrint to specify a default printer for FrameMaker:

<http://www.sundorne.com/FrameMaker/Freeware/setPrint.htm>

SetPrint is included in your **Mif2Go** distribution; see §B [Distribution files](#) on page 1017.

See also:

§36.4.3.1 [Configuring print output: runfm -print and -reverse options](#) on page 984

§36.4.3.2 [Configuring PDF output: runfm -pdf option](#) on page 985

36.4.4 Closing FrameMaker files after conversion

If you specify **-close doc**, **runfm** closes the document file after the conversion is finished; if you specify **-close book**, **runfm** closes the book file. Specify both to close both; FrameMaker remains open, unless you specify **-close all**.

Just as when you run **Mif2Go** interactively, when **runfm** finishes, files that **Mif2Go** opens during conversion (other than files specified by **runfm** options) are closed without saving changes. A book or document file that is already open and active in FrameMaker when **runfm** starts (or is specified by the **-book** or **-doc** option) is either closed without saving or remains open in FrameMaker, depending on the **-close** option.

If you are using **runfm** on the same system as FrameMaker, when you specify **-close all**, **runfm** appends the contents of the FrameMaker console window to the file specified by the **-log** option, before closing FrameMaker; see §36.5.2 [Capturing console diagnostics: runfm -log option](#) on page 988.

If you are using **runfm** across a network, **runfm** does *not* append console messages to a log file when you specify **-close all**; see §36.7 [Operating runfm across a network](#) on page 992.

If you do not specify **-close all**, FrameMaker remains open after **runfm** finishes.

See also:

§36.5.2 [Capturing console diagnostics: runfm -log option](#) on page 988

§36.7 [Operating runfm across a network](#) on page 992

36.5 Troubleshooting runfm processes

To use **runfm**, you must have **Mif2Go** installed on your system.

In this section:

§36.5.1 [Increasing console diagnostics: runfm -diag option](#) on page 988

§36.5.2 [Capturing console diagnostics: runfm -log option](#) on page 988

§36.5.3 [Reviewing FrameMaker console messages after runfm](#) on page 988

§36.5.4 [Troubleshooting failed runfm processes](#) on page 989

36.5.1 Increasing console diagnostics: runfm -diag option

When you use the **-diag** option, **runfm** writes additional diagnostic messages to the FrameMaker console window. These messages record step-by-step details while **runfm** parses the string of options to be passed to `m2rbook.dll`. This option is intended primarily for use by **Mif2Go** developers; however, it can be helpful for working out problems as you construct a **runfm** command sequence.

For conversion runs, the **-diag** option produces the same diagnostics as the following setting (see §35.2 [Activating and logging production of deliverables](#) on page 956):

```
[Automation]
; RunfmDiagnostics = No (default) or Yes (include more diagnostic
; messages in the Frame console file when running from runfm)
RunfmDiagnostics=Yes
```

However, **-diag** is effective for print options (which are handled before **runfm** reads any project configuration file) as well as for conversion options.

See also:

§35.2 [Activating and logging production of deliverables](#) on page 956

§36.5.2 [Capturing console diagnostics: runfm -log option](#) on page 988

§36.5.3 [Reviewing FrameMaker console messages after runfm](#) on page 988

36.5.2 Capturing console diagnostics: runfm -log option

If you are using **runfm** on the same system as FrameMaker, before closing FrameMaker **runfm** appends the contents of the FrameMaker console window to the file specified by the **-log** option; this file is placed in the same directory as the specified book or document. The default log file name is `mif2go.log`. If you specify both **-book** and **-doc**, and the book and document files are in different directories, the log file is placed in the document directory rather than the book directory.

The contents of the FrameMaker console window are appended to the log file only when you specify **-close all** (see §36.4.4 [Closing FrameMaker files after conversion](#) on page 987); if FrameMaker remains open after **runfm** finishes, nothing is appended to the log file.

If you are using **runfm** across a network, the **-log** option has no effect; see §36.7 [Operating runfm across a network](#) on page 992.

Maintaining a log file preserves FrameMaker console output over multiple executions of **runfm**, because FrameMaker clears the console each time it starts. See §36.5.3 [Reviewing FrameMaker console messages after runfm](#) on page 988.

See also:

§36.4.4 [Closing FrameMaker files after conversion](#) on page 987

§36.5.1 [Increasing console diagnostics: runfm -diag option](#) on page 988

§36.5.3 [Reviewing FrameMaker console messages after runfm](#) on page 988

36.5.3 Reviewing FrameMaker console messages after runfm

When you specify **-close all** for the **runfm** command (or for the last in a series of **runfm** commands), provided you are not operating **runfm** across a network (see §36.7 [Operating runfm across a network](#) on page 992), before FrameMaker closes, **runfm** appends the contents of the FrameMaker console window to a text file, default name `mif2go.log` (see §36.5.2 [Capturing console diagnostics: runfm -log option](#) on page 988),

located in the input directory for your project. You should inspect this file for error messages.

If you do not use the `-log` option, after FrameMaker closes (and before you open it again) you can find the console output for the latest FrameMaker session in `consfile.txt`, located in the FrameMaker installation directory.

Some kinds of errors are not reported

The **Mif2Go** *Export* and *Finished* dialogs are suppressed by `runfm`, to make sure **Mif2Go** does not request user intervention during an unattended conversion. If any FrameMaker file that **Mif2Go** tries to access during the conversion (such as a chapter file in a book) has a problem (such as missing fonts or graphics) that would normally result in a dialog, the problem is ignored and the conversion continues. *Such problems are not reported in the FrameMaker console window*; with unattended operation, you have no way to know they occurred. Therefore it is essential to make sure your conversion project is thoroughly debugged before you use `runfm` for unattended operation.

36.5.4 Troubleshooting failed runfm processes

Before you use `runfm` to automate a process, first try the same operation manually, from within FrameMaker. Watch for errors. If the operation does not work manually, `runfm` will not work either!

Note: **Mif2Go** must be installed to use `runfm` successfully.

Get diagnostics and a log file

You can tell `runfm` to write more diagnostics by adding this option to the `runfm` command line:

```
-diag
```

You can save the diagnostic messages to a file with this command-line option:

```
-log logfile.txt
```

New messages are appended to the log file, so you can use the same file for many sessions. If you specify `-log` with no name, **Mif2Go** writes to a file named `mif2go.log`, in the current directory.

PDF output does not always report errors

If you are using `runfm` to produce PDF output, some of the possible errors you can get might not be documented at all for the FDK functions involved, so you will have to do the usual type of troubleshooting for PDF problems. For example, sometimes Acrobat Distiller does not like a particular graphic. If you get an error when you make the PDF manually in FrameMaker (by printing to the Adobe PDF printer, *not* via **SaveAs PDF**), try cutting the file in half, and see which part still has the problem. Repeat until you identify the exact point of trouble.

A missing output file means an error occurred

Acrobat Distiller always writes PDF output to the default Windows document directory for the user who is currently logged in, such as `My Documents` on Windows XP. If Distiller fails, an error message might not be returned from the FDK, and the output file will simply not be present. However, if **Mif2Go** does find the file, **Mif2Go** moves it to the destination you specified.

36.5.5 Running a single Mif2Go conversion or print job

Here are examples of how you can use `runfm` to:

[Convert a book](#)

[Update a chapter](#)

[Convert a single-file document](#)

[Print and update a chapter](#)

Create a PostScript file.*Convert a book*

To open FrameMaker and convert a book, leaving FrameMaker open afterward:

```
runfm -book D:\Guides\UserGuide.book -project "Word for review"
```

To convert a book, with FrameMaker already open and the book file active, then close the book, leaving FrameMaker open:

```
runfm -project "Word for review" -close book
```

Update a chapter

To update a chapter in a book, closing just the chapter file afterward:

```
runfm -book E:\UG.book -doc E:\Ch\Ops.fm -project 4Review -close doc
```

To update a chapter in a book, with the book and the chapter file already open in FrameMaker, and the chapter file active; then close both afterward, leaving FrameMaker open:

```
runfm -project 4Review -close book -close doc
```

Convert a single-file document

To open and convert a single-file FrameMaker document, closing FrameMaker afterward:

```
runfm -doc D:\Guides\ITGuide.fm -project "HTML for IT" -close all
```

To convert a single-file document that is already open in FrameMaker, closing the document afterward but leaving FrameMaker open:

```
runfm -project "HTML for IT" -close doc
```

Print and update a chapter

To print a chapter to Acrobat with the book already open in FrameMaker, then convert the chapter:

```
runfm -doc E:\Ch\Ops.fm -project 4Review -pdf doc E:\2print\Ops.pdf
```

Create a PostScript file

To open a book and print it to a PostScript file in directory My Documents, for later distilling:

```
runfm -book E:\UG.book -print book -reverse no -printer "Adobe PDF"
```

36.5.6 Running a series of Mif2Go conversions

If you do not specify **-close all**, FrameMaker remains open after a **runfm** conversion finishes. This allows you to use a .bat file to run several different conversions of the same book or document, without reloading FrameMaker for each conversion. For example:

```
runfm -book D:\Guides\UserGuide.book -project "Word for review"
runfm -project "On-line help"
runfm -project "HTML version" -close all
```

When you run a series of conversions that use different project configuration files, make sure the configuration files include explicit values for any settings in the following sections with values that differ from one project to the next:

<u>All conversions</u>	<u>HTML-based Help conversions</u>	<u>WinHelp conversions</u>
[Automation]	[CSS]	[HelpContents]
[Setup]	[MSHtmlHelpOptions]	[HelpOptions]
[Graphics]	[JavaHelpOptions]	
	[OmniHelpOptions]	

If two projects have the same configuration settings in these sections (even with different values), or at least if the second project has explicit settings, you should be able to use them in consecutive invocations of **runfm** without closing FrameMaker in between. Otherwise, you risk “bleed-through” of the prior configuration settings. If you find that the second project is not coming out quite right, try running it by itself after closing and reopening FrameMaker.

Here is an example of running multiple conversion projects in the same .bat file:

```
runfm -book E:\UG.book -doc E:\Ch\Ops.fm -project 4Review -close doc
runfm -doc E:\Ch\Examples.fm -project 4Review -close doc
runfm -doc E:\Ch\Glossary.fm -project 4Review -close all -log
runfm -book E:\UG.book -project "On-line help" -close all
runfm -doc D:\Guides\ITGuide.fm -project "HTML for IT" -print doc
runfm -close all -log ITGmsgs.txt
```

This series of commands updates three chapters of the same book in one project; closes FrameMaker (logging console messages for all three to E:\Ch\mif2go.log), then reopens FrameMaker to convert the same book using a different project; then closes FrameMaker again (logging console messages to E:\mif2go.log), and reopens it to convert a single-file document using yet another project, also printing the document directly to the current printer; then shuts down FrameMaker, logging console messages for the last project to D:\Guides\ITGmsgs.txt.

Because FrameMaker remains open after the first conversion, it is not necessary to repeat the **-book** option for the second and third conversions. However, after closing FrameMaker, the **-book** option is needed again for the fourth conversion.

36.5.7 Including runfm in a multi-step or scheduled process

Because **runfm** can be invoked in a .bat file, you can include **runfm** commands interspersed with other commands. For example, we use BuildUG.bat to prepare all editions of the **Mif2Go User's Guide** for release. The first part of the script generates and archives the WinHelp edition, then uploads the archive to a server (the **runfm** command and its arguments actually are all on one line):

```
rem Usage: BuildUG NN
rem       where NN is the two-digit release number.
@echo off
if "%1" == "" goto NOARG
if not exist G:\OmniSys\UG\source\history.txt goto NOHIST
cd G:\OmniSys\UG\out
echo BuildUG %1 starting %DATE% at %TIME% > ug33v%1.log
copy /Y /V G:\OmniSys\UG\source\history.txt G:\OmniSys\UG\out
copy /Y G:\OmniSys\UG\cfg\*.ini G:\OmniSys\UG\cfg\CFGbackup\*.i%1
:WINHELP
runfm -book G:\OmniSys\UG\usergd.book -project WinHelp
      -close all -log ug33v%1.log > ug33v%1.log 2>&1
if not exist G:\OmniSys\UG\hlp\ugmif2go.hlp goto NOWIN
copy /Y G:\OmniSys\UG\hlp\*.ini G:\OmniSys\UG\hlp\WHbackup\*.i%1
copy /Y ftpug.txt ftp%1.txt
echo send ughlp%1.zip >> ftp%1.txt
echo bye >> ftp%1.txt
ftp -i -s:g:\omnisis\ug\out\ftp%1.txt
echo ughlp%1.zip finished uploading at %TIME% >> ug33v%1.log
. . .
```

You can include **runfm** commands in a .bat file that you set up as a Windows Scheduled Task. For example, you could use **runfm** to periodically output MIF versions of all the files in a book, then process the MIF files with another application, such as an archive or index utility. In fact, you could use just the evaluation version of **Mif2Go** to produce MIF output via **runfm**.

36.6 Comparing runfm with the DCL command-line filter

With **runfm** you can do any of the following:

- Start FrameMaker from the command line, and have **Mif2Go** (or some other FrameMaker plug-in) invoked automatically.
- With FrameMaker already open and a book or document selected, run a **Mif2Go** conversion from the command line.
- Run multiple conversions, with a series of **runfm** commands in a Windows **.bat** file.
- Optionally close FrameMaker (or the book or document or both) automatically when all conversions are complete.

*Advantages of
runfm*

The advantage of **runfm** over the DCL command-line filter is availability of all the **Mif2Go** options that are excluded from DCL command-line operation, including book conversion, template import, and postprocessing steps; in other words, if you want to do any of the following:

- convert a FrameMaker book
- generate bitmap graphics
- automatically import formats from FrameMaker templates
- automatically create and delete **.mif** files
- automatically create configuration files
- automatically create Help-system project files.

See §37.1 [How the DCL filter works](#) on page 995.

*Advantages of
DCL*

However, with DCL you do not need FrameMaker at all (just the MIF files to be converted), and you can use a Windows **.bat** file to execute system commands before and after conversion. Also, DCL is faster, if all you are doing is including documentation as part of a build process, after writers are satisfied with the results of interactive conversions. See §37 [Converting via DCL](#) on page 995.

36.7 Operating runfm across a network

To use **runfm** on a network with FrameMaker on a different system, you must observe the following restrictions:

- the system where FrameMaker and **Mif2Go** are installed must be in the same domain as the system where you are using **runfm**
- you must be logged into both systems with the same user ID
- FrameMaker must be already open on the remote system.

*Specify **-remote**
for network
operation*

To operate **runfm** across a network, specify **-remote** with the name of the remote system where FrameMaker is installed. The name to use for the remote system is the name you see under **My Network Places** in Windows Explorer. Omit any punctuation.

*Use FrameMaker
CLSID for
-progid*

For **-progid** you must supply the automatic FrameMaker **progid**, which is the CLSID (class identifier) found in the remote-system Windows Registry for the following key:

```
HKEY_LOCAL_MACHINE/Software/Classes/FrameMaker.Api
```

Enclose the CLSID in braces; for example:

```
-progid {539DB5D0-C0C6-11D0-985E-0060970BEC0B}
```

*No log file with
remote operation*

When you specify **-remote** you can use the **-close** option, but if you **-close** all, **runfm** does *not* copy the FrameMaker console output to a log file (see §36.5.3 [Reviewing FrameMaker console messages after runfm](#) on page 988).

*Set up both
systems for
DCOM*

For remote operation you might have to specify system settings to enable DCOM (Distributed Component Object Model) on both machines. On Windows XP Pro (for example), type **dcomcnfg** at a command prompt and press Enter; the Component Services console opens. Navigate to **Component Services > Computers > My Computer >**

DCOM Config, and select **FrameMaker API**. On the **Action** menu choose **Properties**; the *FrameMaker API Properties* dialog opens. For information about the settings available, try **Help**. For additional information, see *Running asynchronous clients on remote hosts* in the **FDK Platform Guide**, `winguide.pdf`.

36.8 Using runfm for other FrameMaker plug-ins

You can use **runfm** with different **-client** options to run FrameMaker plug-ins other than **Mif2Go**, provided you know the values for the following options:

- client** the `ClientName` of the plug-in
- project** the text the plug-in expects when notified of an FDK `ClientCall`.

The command would be like this:

```
runfm -client TheClientName -project "The expected text"
```

You can use Windows Explorer to find the `ClientName` for a plug-in. In the FrameMaker plug-ins directory select the DLL file name, go to **File > Properties**, and choose the **Version** tab. The `ClientName` should be the first item under **Other version information**.

You can use the same **-progid** option you specified when you set up FrameMaker for **Mif2Go**; this option works for any plug-in. You can use the **-book** and **-doc** options to specify the files you want open; these files are opened before the plug-in is called. You can use the **-close** option the same way as for **Mif2Go**.

You do not need the **-client** option for **Mif2Go** projects; this option is intended only for use with other FrameMaker plug-ins. The default value is `OmniBookExport`, which is the `ClientName` for **Mif2Go** DLL file `m2rbook.dll`.

See also:

§36.2 [Setting up FrameMaker for unattended operation](#) on page 980

37 Converting via DCL

Conversion from a command line, using the DCL (Document Coding Language) filter, is intended for programmers adding **Mif2Go** to automated build systems. DCL is run separately from FrameMaker; the DCL filter operates only on FrameMaker MIF (Maker Interchange Format) files.

Note: If any graphics are embedded in your document, or if any graphics contain FrameMaker vector elements (such as callouts); or if your FrameMaker files are binary rather than MIF; **Mif2Go** must be run from within FrameMaker. See §34.3 [Considering ways to automate conversions](#) on page 937.

This section shows how to operate the **Mif2Go** DCL filter. Topics include:

- §37.1 [How the DCL filter works](#) on page 995
- §37.2 [Using the DCL filter](#) on page 996
- §37.3 [DCL command-line syntax](#) on page 998
- §37.4 [Command-line examples](#) on page 1000
- §37.5 [Converting in multiple steps via DCL](#) on page 1002
- §37.6 [Specifying output file paths and names](#) on page 1002
- §37.7 [About DCL technology](#) on page 1003

See also:

- §36 [Converting via runfm](#) on page 979
- §38 [Generating intermediate output](#) on page 1005

37.1 How the DCL filter works

The command-line version of **Mif2Go** provides a restricted set of features, and is intended *only* to support automated build systems. This command-line method assumes you have already set up your project, using the FrameMaker plug-in; and that further conversion runs do *not* need to do any of the following:

- convert a FrameMaker book
- generate bitmap graphics
- automatically import formats from FrameMaker templates
- automatically create and delete .mif files
- automatically create configuration files
- automatically create Help-system project files.

You *must* run **Mif2Go** from within FrameMaker to accomplish any of the above. To set up a command-line system that will handle these requirements, see §36 [Converting via runfm](#) on page 979.

Mif2Go uses the DCL (Document Coding Language) filter to convert MIF files according to settings you have already specified in a configuration file (and optionally as arguments to the DCL command). Before you can convert files this way, you must do the following:

- Create a configuration file for the conversion, using the **Mif2Go** FrameMaker plug-in and a text editor such as Notepad.
- Save as MIF all the FrameMaker files that are to be converted (see §38 [Generating intermediate output](#) on page 1005).

When your configuration file and MIF files are ready, you run the **Mif2Go** DCL filter at a command-line prompt in a command window.

An advantage of using **Mif2Go** this way is that you can automate batch processing of many files. The **Mif2Go** DCL filter is usable even if you do not have FrameMaker, and were given FrameMaker-generated MIF files by someone else. However, the filter cannot process MIF files generated by programs other than FrameMaker. If you want to process such files, load them first in FrameMaker and save them as MIF, then process those MIF files.

37.2 Using the DCL filter

Command-line conversion is intended solely for use in automated build systems, where the process has been worked out first interactively, via FrameMaker; see §3 [Converting a book or document](#) on page 77.

In this section:

- §37.2.1 [Understanding where to run DCL](#) on page 996
- §37.2.2 [Preparing for conversion](#) on page 996
- §37.2.3 [Converting a single MIF or DCL file](#) on page 996
- §37.2.4 [Converting a group of MIF or DCL files](#) on page 997
- §37.2.5 [Merging ancillary Help files with DCL](#) on page 997

37.2.1 Understanding where to run DCL

You must invoke the **Mif2Go** DCL filter on a command line in a Windows *Command Prompt* window. The **Mif2Go** DCL filter is a Windows Console application, not an MS-DOS application. It will not run under plain MS-DOS, without Windows.

37.2.2 Preparing for conversion

Before you use the DCL command-line method to convert files, you must do the following:

1. Save the files to be converted in MIF format: on the FrameMaker **File** menu, select **Save As....** For **Save as type** choose MIF (.mif).
2. Copy a starting configuration file for the output type you want (see [Table 30-5](#) on page 859) from %OMSYSHOME%\m2g\local\config to the same directory where you saved the MIF files.
3. Optionally, create a document information file (see §37.4.1 [Creating a document information file](#) on page 1001). Though not strictly necessary, it is often useful to have this information before adjusting configuration settings.
4. Edit the configuration file to specify settings. See §4.1 [Working with Mif2Go configuration files](#) on page 91 for more information.

37.2.3 Converting a single MIF or DCL file

To convert a single FrameMaker MIF file with the **Mif2Go** DCL filter:

1. Open a Windows *Command Prompt* window.
2. Change to the directory where you saved the MIF file and placed a configuration file.
3. At the command-line prompt, enter the following command:

```
dcl -f format [-o output] input.mif
```

where the arguments are as follows:

<i>format</i>	Output type; one of the format codes or names listed for -f in §37.3.1 Command-line switch -f format on page 998.
<i>output</i>	Name or extension (with leading period) of the file to be produced; optional for DCL, RT,F or HTML output, required for XML output.
<i>input</i>	Name of the file to be converted.

4. Press **Enter** to convert the file.

37.2.4 Converting a group of MIF or DCL files

You cannot use the **Mif2Go** DCL filter to convert a book file *per se* from the command line. However, you can convert more than one file at a time (such as all the files *in* a book), by using wildcards in file names, or by executing the **Mif2Go** DCL filter in a batch file. You must make sure any path values in the [Setup] section of the configuration file are correct. See §37.4.3 [Converting a group of files to RTF](#) on page 1001 for an example of using a batch file.

When you convert multiple files, you must provide `mif2go.ini`, which contains file identifiers for the output files; see §5.3.4 [Working with Mif2Go FileIDs](#) on page 119.

37.2.5 Merging ancillary Help files with DCL

You can use DCL to merge contents, index, and other data files for HTML-based Help, by providing a file that lists the items to be included. For example, for HTML Help you can use DCL to merge the part files (`.bh*`) to make a TOC or index; for OmniHelp, you can merge the full-text search and context-sensitive help files.

The list file must have extension `.lst`. The first line in this file must begin with `LIST`, and each subsequent line must specify the base name of a FrameMaker file to be included, with a special file-name extension. The extension must be one of the following:

<code>chp</code>	FrameMaker chapter file
<code>frm</code>	FrameMaker LOF, LOT, or LOR
<code>toc</code>	FrameMaker TOC
<code>idx</code>	FrameMaker standard IX
<code>bkm</code>	FrameMaker special index (IOM or IOR)
<code>gen</code>	FrameMaker generated file other than the above

For example, list file `M2GTest.lst` might look like this:

```
LIST for C:\test\m2gts\hh\M2GTest.lst
Cover.chp
M2GTestLOF.frm
M2GTestLOT.frm
Intro.chp
TextFmts.chp
FigsTbIs.chp
OtherFmts.chp
Appendix.chp
```

For this example the DCL command, which must be executed in the directory where the list file is located, would look like this for HTML Help:

```
dcl -fMB M2GTest.lst
```

See §7.3.4.3 [Merging contents and index files from the command line](#) on page 208.

37.3 DCL command-line syntax

A `dcl` command has the following syntax:

```
dcl [-f format] [-o output] input ... [-v]
```

Command-line switches and arguments override corresponding configuration-file settings. Switches can appear in any order preceding the name(s) of the input file(s) to which they apply. Switches should be lowercase, and a space is required between a switch and its argument. For example:

```
-f HTML
```

Each switch affects only the *input* files named after it on the command line.

In this section:

§37.3.1 [Command-line switch -f format](#) on page 998

§37.3.2 [Command-line switch -o output](#) on page 999

§37.3.3 [Command-line argument input ...](#) on page 999

§37.3.4 [Command-line switch -v](#) on page 1000

§37.3.5 [Additional command-line switches](#) on page 1000

37.3.1 Command-line switch -f format

The DCL `-f` switch specifies the output format, with an optional suffix that generates additional processing for certain formats.

The `-f` options have the following meanings:

<u>Format name</u>	<u>Optional suffix</u>	<u>Description</u>
HTML	F L	HTML 4.0
XHTML	F L	XHTML 1.0
HTMLHelp	F L C I B	Microsoft HTML Help
JavaHelp	F L C I B	JavaHelp
OracleHelp	F L C I B	Oracle Help for Java
EclipseHelp	F L C I B	Eclipse Help
OmniHelp	F L C I B	Cross-platform OmniHelp
DITA	F L	DITA XML
DocBook	F L	DocBook XML
XML	F L	Generic XML
Word	F L	Word 8/97
7	F L	Word 7/95
P <i>or</i> WordPerfect	F L	WordPerfect
WinHelp	F L	WinHelp 4
3	F L	WinHelp 3
F		FrameMaker MIF only

Run wrap-and-ship tasks only

Where F and L are listed under **Suffix**, to run only wrap-and-ship tasks for a project you can append one of these letters to the format name (case does not matter):

F - For a single-file project, run just the wrap-and-ship tasks

L - For a book-level project, run just the wrap-and-ship tasks

You can use dummy names for the input, because wrap-and-ship tasks are independent of the content files. For example:

```
dcl -f htmlL StartAuto.mif
```

See §34 [Automating Mif2Go conversions](#) on page 933 and §35 [Producing deliverable results](#) on page 955.

*Merge contents
and/or index*

These suffix options apply to HTML-based Help systems for which **Mif2Go** can generate contents and/or index. For example: **-f HTMLHelpB**.

C - Merge contents

I - Merge index

B - Merge both contents and index

Letters C, I, and B represent three mutually exclusive options for the same merge operation, which can also (re-) create the project file (depending on configuration settings) while generating the other infrastructure files for the designated Help system.

Note: When you specify suffix C, I, or B for the **-f** argument, the input file must have extension `.lst`; see §37.3.3 [Command-line argument input ...](#) on page 999.

37.3.2 Command-line switch **-o output**

The DCL **-o** switch specifies an output file name (with or without path), or an output file extension:

- o file** *Output file path or name, without extension.* Applies only to the first *input* file name that follows this option. Overrides, for the next file name only, any **-o .ext** or **-o path** that appears earlier on the command line. The default is the same name as the *input* file name, but with the `.ext` extension provided as an argument to an earlier **-o** switch.
- o .ext** *Output file extension, with leading period.* Overrides default output file extensions. Applies to all following *input* file names on the command line until **dcl** encounters a new **-o .ext**. If no output type is specified via **-t** (see §37.3.5 [Additional command-line switches](#) on page 1000), the default value for `.ext` depends on the value of the **-f** argument (see §37.3.1 [Command-line switch -f format](#) on page 998). The value must have a period as the first character. If the period is not present, *ext* is interpreted as a file name.

Default file extensions are as follows:

<u>Output type</u>	<u>Default extension</u>
HTML	.htm
RTF	.rtf
DCL	.dcl
XML	<i>varies</i>

XML is the only output type where you *must* specify **-o .ext**. Otherwise, some of your output files might get extension `.htm`.

37.3.3 Command-line argument *input ...*

The *input ...* argument(s) specify input file name(s) or complete path(s); wildcards are acceptable. If a path or file name contains spaces, surround it with double quotes; for example:

```
dcl -f HTML "C:\My Documents\some.mif"
```

When you specify suffix C, I, or B for the **-f** switch, the input file must have extension .lst; see §37.3.1 [Command-line switch -f format](#) on page 998.

37.3.4 Command-line switch -v

The **-v** switch produces verbose output; **dcl** reports, at the command prompt, everything it does.

37.3.5 Additional command-line switches

Additional switches are available for DCL. You would need these only for working with intermediate DCL input and output formats:

```
dcl [-ab] [-lm] [-s source] [-t target]
```

[Table 37-1](#) shows the options for these additional switches.

Table 37-1 DCL intermediate input and output options

Switches	Purpose	Value	Description
[-ab]	Type of DCL output file	-a	ASCII
		-b	Binary
[-lm]	Endianness of input files	-l	Little-endian (Intel)
		-m	Big-endian
[-s <i>source</i>]	Type of input file	-s	dcl, dcb, mif, lst, or xml
[-t <i>target</i>]	Type of output file	-t	dcl, dcb, inf, rtf, htm, or xml

Types of input or output files for switches **-s** and **-t**:

dcb	Intermediate Document Coding Language binary format; see §37.7.1 DCL file structure on page 1003
dcl	Intermediate Document Coding Language ASCII format; see §37.7.1 DCL file structure on page 1003
htm	HTML, HTML-based Help, XML
inf	Intermediate document information format; see §37.4.1 Creating a document information file on page 1001
lst	Mif2Go -generated list file; see §C.2.3 Additional conversion files on page 1021.
mif	Maker Interchange Format (FrameMaker)
rtf	Rich Text Format
XML	DITA, DocBook, generic XML

37.4 Command-line examples

This section presents the following **Mif2Go** conversion examples:

- §37.4.1 [Creating a document information file](#) on page 1001
- §37.4.2 [Writing converted files to a different directory](#) on page 1001
- §37.4.3 [Converting a group of files to RTF](#) on page 1001
- §37.4.4 [Converting a file to HTML](#) on page 1001
- §37.4.5 [Converting from one DCL format to another](#) on page 1001
- §37.4.6 [Generating DITA output via command line](#) on page 1002

37.4.1 Creating a document information file

A *document information file* is an ASCII file with extension `.inf` that lists all fonts, formats, reference frames, text flows, cross-referenced files, and imported graphics used in a FrameMaker document. To create a document information file for `filename.mif`, run the **Mif2Go** DCL filter with the following options:

```
dcl -t inf filename.mif
```

This command creates file `filename.inf` in the same directory as `filename.mif`.

37.4.2 Writing converted files to a different directory

To convert files `ch*.doc`, `title.doc`, and `TOC.doc` to RTF with file extension `.new` and place the converted files in directory `c:\myfiles\newstuff`:

```
dcl -o c:\myfiles\newstuff -o .new ch*.doc title.doc TOC.doc
```

Note: You do not have to use `.mif` as the source file-name extension; however, the source file must be in MIF format.

37.4.3 Converting a group of files to RTF

If you have a group of FrameMaker MIF files, for example `ch*.mif`, this command converts them to RTF for import into Microsoft Word:

```
dcl ch*.mif
```

Although you can use wildcards in the *source* argument or list multiple *source* files on the command line, **Mif2Go** might run out of memory if there are many files, or if the files are complex. Instead, you can use one of the following equivalent methods.

- On the command line:

```
for %f in (*.mif) do dcl "%f"
```

- In a `.bat` file:

```
for %%f in (*.mif) do dcl "%%f"
```

Note: Double quotes are *required* around the **dcl** *source* argument if any file names or paths contain spaces.

37.4.4 Converting a file to HTML

To convert file `myfile.mif` from MIF to standard HTML:

```
dcl -f HTML -t htm myfile.mif
```

The output HTML file is created in the same directory as `myfile.mif`, and is named `myfile.htm`.

Note: If you specified CSS output in your project configuration file, you will also see a file with extension `.css`, which you can edit as plain text. Not all browsers support CSS; some might ignore the `.css` files.

37.4.5 Converting from one DCL format to another

You can use the **dcl** program to convert from one form of DCL to the other. ASCII DCL files normally have extension `.dcl`, while binary DCL files have extension `.dcb`.

To convert an ASCII DCL file to binary DCL:

```
dcl -t dcb myfile.dcl
```

To convert a binary DCL file to ASCII DCL:

```
dcl -t dcl myfile.dcb
```

To generate ASCII DCL directly from a MIF file, use target option dcl:

```
dcl -t dcl myfile.mif
```

To generate an RTF file from DCL, you can omit the target option:

```
dcl myfile.dcb
```

The **dcl** program uses **dcx.dll** to convert from one form of DCL to the other.

37.4.6 Generating DITA output via command line

After you have set up a DITA project, you can run the conversion from the command line (see §37 [Converting via DCL](#) on page 995), as follows:

1. Run the conversion first from within FrameMaker, via **File > Save using Mif2Go**. On the *Export* dialog, replace the value of **Suffix for created files:** with **.dcl**. This will produce a *filename.dcl* file for each FrameMaker file (see §3.7.2 [Specifying output type and file extension](#) on page 84).
2. Open a Windows *Command Prompt* window and **cd** to your project directory, which now contains one or more **.dcl** files, along with your project configuration file.
3. Type the following command at the command prompt, then press Enter:

```
dcl -t htm -f DITA -o .dita filename.dcl
```

where *filename* is the name of one of the FrameMaker files you converted. (See §37.2.4 [Converting a group of MIF or DCL files](#) on page 997 for batch conversions.)

This process will produce a new *filename.dita*. If the process appears to be taking more than a few seconds, terminate it with **Ctrl+C**.

If you modify anything in a FrameMaker file, you will have to start from [Step 1](#) again and regenerate the **.dcl** for that file. Otherwise, you can iteratively change settings in the configuration file and rerun from the command line until you are satisfied with the result.

37.5 Converting in multiple steps via DCL

To export graphics, modify them, and then continue the conversion using the modified graphics, you must run **dcl** twice. For the first step, specify target option **-t dcl**, which tells the filter to stop after producing the DCL file and exported graphics files. For the second step, specify the **.dcl** file as the source; you do not need source option **-s dcl**, because the filter can tell both from the extension on the DCL file and from its opening bytes that the source file is in DCL format instead of MIF.

37.6 Specifying output file paths and names

For the output file name, you can modify any or all of the path, name, and extension. By default, the filter alters only the file extension. For RTF output, the extension is normally **.rtf**. For multi-step processing, it is **.dcl** for the first step and **.rtf** for the last step. The target file is written to the same directory as the source file, usually the current directory. Any intermediate files (typically binary DCL files, **.dcb**) are written to the current directory, and are automatically deleted after conversion is complete.

The output option **-o name** can specify a path without a file name, a file name with or without a path, or an extension without a file name. Each of these works differently:

- **Path without file name** causes the output file to be written with the same name but to a different directory.
- **File name with or without path** alters the file name for the output file. If you do not specify a path, the original file path (as modified by any earlier path-related `-o` option) is used.
- **Extension without file name** gives the output file the extension specified instead of the original extension. (In some cases, the new extension is added on instead of replacing the previous one; this happens if the previous extension was *not* the one used to indicate the input format, such as `.mif`, and if the file naming rules for the system permit multiple extensions.)

37.7 About DCL technology

The **Mif2Go** DCL filter is based on the Omni Systems Document Coding Language, **DCL**. This section gives a brief overview of DCL. For a full description of DCL, see the Omni Systems *DCL Specification*, available on request. Omni Systems has placed the DCL language in the public domain; you may use it without obligation. Omni Systems products based on DCL, such as **Mif2Go**, are proprietary, and must be licensed from Omni Systems.

In this section:

§37.7.1 [DCL file structure](#) on page 1003

§37.7.2 [Writing DCL conversion modules](#) on page 1003

37.7.1 DCL file structure

DCL can be read and written in either of two formats: ASCII or binary. When the **Mif2Go** DCL filter is converting your files, it writes and reads the binary form, which is designed for very rapid and efficient processing. If you want to work with the DCL file yourself, use the ASCII version, which can be edited in any plain-text editor. All Omni Systems DCL programs understand both forms of DCL; for example, `drmi f` can write either format, and `dwr t f` can read either format.

37.7.2 Writing DCL conversion modules

For simple projects, you can use text-processing tools to modify ASCII DCL files. You can search and replace format names, for example, or modify format properties.

For more complex projects, where you need the power and versatility of a full-sized programming language such as perl, Java, or C++, you are better off working with binary DCL. You write a program that reads binary DCL files. Your program reads the eight-byte “controls” in a binary DCL file one at a time; when it has read one control, your program knows immediately how much “external” data follows the control, which tells it where the next control begins. This design makes it simple for a program to step to the specific controls it needs to modify. Once there, your program can replace or delete the control, or add more controls, without concern for side effects elsewhere.

The Omni Systems DCL programs are written in C++, using a portable class library developed by Omni Systems. If you intend to write C++ programs that work with DCL, ask Omni Systems about availability of sample code and development tools.

38 Generating intermediate output

If you run a **Mif2Go** conversion in two stages, stopping the first stage as soon as MIF or DCL files have been created (see [Figure 1-1](#) on page 62), you can use the intermediate files for other purposes; also, you can set up a script to modify the files and then run **Mif2Go** again to complete the conversion.

This section presents configuration settings for producing FrameMaker MIF or ASCII DCL files from your FrameMaker document. Topics include:

§38.1 [Producing MIF with Mif2Go vs. FrameMaker](#) on page 1005

§38.2 [Generating MIF output](#) on page 1006

§38.3 [Converting to ASCII DCL](#) on page 1009

§38.4 [Generating ASCII DCL output](#) on page 1011

38.1 Producing MIF with Mif2Go vs. FrameMaker

When you specify FrameMaker MIF as the output type, **Mif2Go** saves all the files in your FrameMaker document (including the book file, if you so specify) in MIF format, and stops there.

Use this feature to save your entire document as MIF, perhaps to a different directory, and perhaps for a different purpose; for example:

- Store your FrameMaker document in a revision-control system.
- Pass files to someone using FrameMaker on a different operating system.
- Extract text for translation.
- Back up your document over a LAN (Local Area Network).

*Use Wash option
instead to clean
files*

If you want to save as MIF primarily to clean up FrameMaker file corruption, you can simply use the **Wash Via MIF** option on the FrameMaker File menu; see §D.2.6 [Check for file corruption](#) on page 1032.

*FrameMaker
insists on .mif
extension*

When you save as MIF from FrameMaker, you have to use the `.mif` extension. But if you are saving a book file, you do not really want it to come out as `mydoc.book.mif`, or worse yet, `mydoc.mif`. And if you subsequently load chapters into FrameMaker from the book file, the extension is wrong; the book knows about `chapter.fm`, not `chapter.mif`.

*FrameMaker 8
defaults to
version 7 MIF!*

When an FDK application saves as MIF, the default for FrameMaker version 8 is to produce *FrameMaker 7* MIF files. This default action discards the FrameMaker-8-specific information, and does not use Unicode. **Mif2Go** overrides this action, and saves FrameMaker 8 files as FrameMaker 8 MIF, by default. See §38.2.4 [Saving FrameMaker 8 files as FrameMaker 8 MIF](#) on page 1008.

*Mif2Go keeps
original file
extensions*

Mif2Go produces MIF files with `.book` and `.fm` extensions, saved in a different directory. When you load these files into FrameMaker, FrameMaker reads the MIF without complaining; and when you choose **File > Save**, FrameMaker silently overwrites them in binary format.

*FrameMaker
changes links*

When you save to a different directory in FrameMaker, FrameMaker thoughtfully rewrites all your links to point back to the original sources.

*Mif2Go
preserves links*

Mif2Go first saves each MIF file in the same directory as the original file, but with a temporary extension; then moves the file to the new directory, renaming it with a `.book` or `.fm` extension.

See also:

§38.2 [Generating MIF output](#) on page 1006

§D.2.6 [Check for file corruption](#) on page 1032

38.2 Generating MIF output

In this section:

§38.2.1 [Understanding how MIF files are generated](#) on page 1006

§38.2.2 [Setting up a FrameMaker MIF project](#) on page 1006

§38.2.3 [Specifying which files to include in MIF output](#) on page 1007

§38.2.4 [Saving FrameMaker 8 files as FrameMaker 8 MIF](#) on page 1008

§38.2.5 [Saving FrameMaker 9+ files as FrameMaker 7 MIF](#) on page 1008

§38.2.6 [Specifying file extensions for MIF output](#) on page 1008

38.2.1 Understanding how MIF files are generated

The **Mif2Go** MIF output type was designed to work with CVS revision-management software, to store full FrameMaker books. Special features make it possible to seamlessly reload the stored files into FrameMaker:

- MIF output files retain the same paths as the `.fm` files, rather than receiving new paths relative to the new location. This is different from FrameMaker **File > Save As...** behavior, which alters paths; a feature that can make reloading MIF files a nightmare.
- MIF output files are named with an `.fm` extension, so they load from the MIF (or native FrameMaker) book file.
- You can use a FrameMaker conversion template (see §2.4 [Importing formats from a conversion template](#) on page 67) to alter properties such as format definitions in the MIF output files, without affecting the original FrameMaker files.

Mif2Go first creates the MIF files in the same directory with your FrameMaker document files, using temporary file extensions; then moves the MIF files to the project directory, and changes the file extensions to whatever you specify. This two-step process accomplishes the following:

- Prevents FrameMaker from converting relative references (such as paths to external graphics files) to absolute references.
- Prevents FrameMaker from creating `.backup` files in the project directory. Any previous copies in the project directory are overwritten by the move.

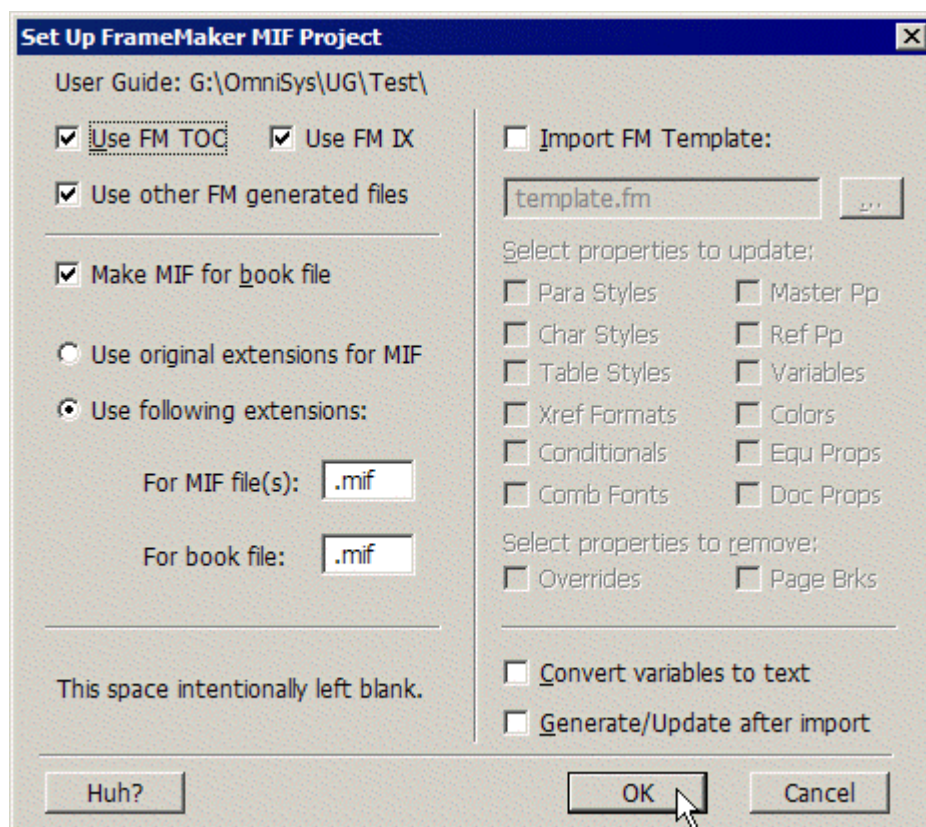
Mif2Go does not support FrameMaker books that contain chapter files in other directories. MIF files are always written to a single output directory.

38.2.2 Setting up a FrameMaker MIF project

When you select FrameMaker MIF as the output type for a new project, the *Set Up* dialog shown in [Figure 38-1](#) opens. [Table 38-1](#) shows the corresponding settings in the configuration file. When you specify FrameMaker MIF as the output type, **Mif2Go** saves all the files in your FrameMaker document (including the book file, if you so specify) in MIF format, and stops there.

See also:

§3.4 [Choosing project set-up options](#) on page 79

Figure 38-1 Set Up FrameMaker MIF Project**Table 38-1 FrameMaker MIF set-up options and configuration settings**

Set-up dialog Option	Configuration file Section	Setting	Default	Ref.
Make MIF for book file	[Setup]	MakeBookMIF=Yes	Yes	38.2.3
Use original extensions for MIF	[Setup]	OrigExtForMIF=Yes	Yes	38.2.6
Use following extensions:	[Setup]	OrigExtForMIF=No	Yes	38.2.6
For MIF file(s)		FileSuffix=.any	.mif	
For book file		MIFBookSuffix=.any	.mif	

Note: When you click **OK** to dismiss the FrameMaker MIF *Set Up* dialog, and Notepad opens to show configuration settings based on your choices, *do not be alarmed* if you see settings that purport to make both file extensions .mif, even though you specified original extensions. A later setting overrides the earlier settings.

38.2.3 Specifying which files to include in MIF output

You can specify whether to save the book file itself as MIF:

```
[Setup]
; Settings for FrameMIF projects
; MakeBookMIF = Yes (default, make MIF of book file too) or No
MakeBookMIF=Yes
```

You might want generated files to be saved as MIF, or you might not. You can choose individually for contents and index files; all others are included or excluded as a group:

```
[Setup]
; UseFrameTOC = Yes (default, exclude for Help formats), or No
UseFrameTOC=No
```

```

; UseFrameIX = Yes (default, exclude for Help formats), or No
UseFrameIX=No
; UseFrameGenFiles = Yes (default for all formats) or No
UseFrameGenFiles=No

```

See §5.5 [Converting FrameMaker-generated files](#) on page 124 for more information.

38.2.4 Saving FrameMaker 8 files as FrameMaker 8 MIF

By default, when you save as MIF directly from FrameMaker 8, you get FrameMaker 7 MIF files, losing Unicode capability and possibly other FrameMaker 8 features. However, **Mif2Go** saves your FrameMaker 8 files as FrameMaker 8 MIF *by default*, preserving Unicode and other features.

To save FrameMaker 8 files as FrameMaker 7 MIF:

```

[Setup]
; UseFrame8MIF = Yes (default, for Frame 8 only),
; or No (to get Frame 7 MIF)
UseFrame8MIF = No

```

This setting takes effect only if you are using FrameMaker version 8.

38.2.5 Saving FrameMaker 9+ files as FrameMaker 7 MIF

By default, when you save as MIF directly from FrameMaker 9, you get FrameMaker 9 MIF files. You cannot get FrameMaker 8 MIF files from FrameMaker 9; however, FrameMaker 9 MIF is almost identical to FrameMaker 8 MIF. And **Mif2Go** *can* force output of FrameMaker 7 MIF files from FrameMaker 9.

To save FrameMaker 9 (or 10, or 11...) files as FrameMaker 7 MIF:

```

[Setup]
; UseFrameXMIF = Yes (default), or No (to get Frame 7 MIF),
; where x is 9, 10, 11, or a later version number.
UseFrame9MIF = No

```

We do not recommend this setting unless you must supply MIF files to someone using FrameMaker 7. This setting takes effect only if you are using FrameMaker version 9 or a later version.

38.2.6 Specifying file extensions for MIF output

You can specify file extensions separately for the book file and for all other files; the default in either case is to retain the original FrameMaker file extension. For example:

```

[Setup]
; Settings for FrameMIF projects
; MakeBookMIF = Yes (default, make MIF of book file too) or No
MakeBookMIF=Yes
; OrigExtForMIF = Yes (default, use original FM file extensions) or No
OrigExtForMIF=No
; MIFBookSuffix = suffix to use for book file MIF, default ".mif"
MIFBookSuffix=.book
; FileSuffix = suffix to use for MIF files, default ".mif"
FileSuffix=.fm

```

If you leave file extensions alone, all the files in your document are saved with default extension `.mif`. You can specify different extensions, depending on the reason for saving your document as MIF.

If you specify `OrigExtForMIF=Yes`, **Mif2Go** saves all the files in your document in MIF format, but with their original FrameMaker file extensions: one extension for the book file (provided you also specify `MakeBookMIF=Yes`), and one for the files contained in the book.

If you specify `OrigExtForMIF=No`, you can specify one extension for the book file (provided you also specify `MakeBookMIF=Yes`), and another extension for the files contained in the book.

Why would you want MIF files to have FrameMaker extensions? Suppose you are saving as MIF so you can store your original document in a revision-control system such as CVS. You cannot just check binaries into CVS if you ever want to compare revisions using the CVS `diff` function, which is a huge benefit. You must store files in an ASCII format, so that CVS can provide metadata. You have these choices:

- Name the files `*.mif`, and rename them one by one every time you copy them into or out of the archive, from or to your working directory.
- Keep the original FrameMaker file extensions even though the files are in MIF format, so you can copy all the files to your working directory and open them in FrameMaker without changing the extensions.

You might want to specify different extensions if you are passing FrameMaker files back and forth with someone who is using FrameMaker on a different system with different file-naming conventions.

38.3 Converting to ASCII DCL

When you specify ASCII DCL as the output type, **Mif2Go** saves all the files in your FrameMaker document in MIF format, and then converts the MIF files to ASCII DCL files (with extension `.dcl`) instead of the usual binary DCL files (with extension `.dcb`), and stops there.

You can use this feature to export embedded graphics from your FrameMaker document so you can convert them outside of **Mif2Go**, or replace them, before you continue the conversion; see §5.7.3.2 [Processing embedded graphics separately](#) on page 132.

You can use Perl scripts to extract information from the DCL files to integrate with other documents from non-FrameMaker sources. Perl can parse DCL files much easier than it can parse MIF files.

You can also create DCL files, modify them to remap something on the way from FrameMaker to RTF or HTML, then run **Mif2Go** again to finish the conversion, starting from the modified DCL files.

For more information, see:

§38.4 [Generating ASCII DCL output](#) on page 1011

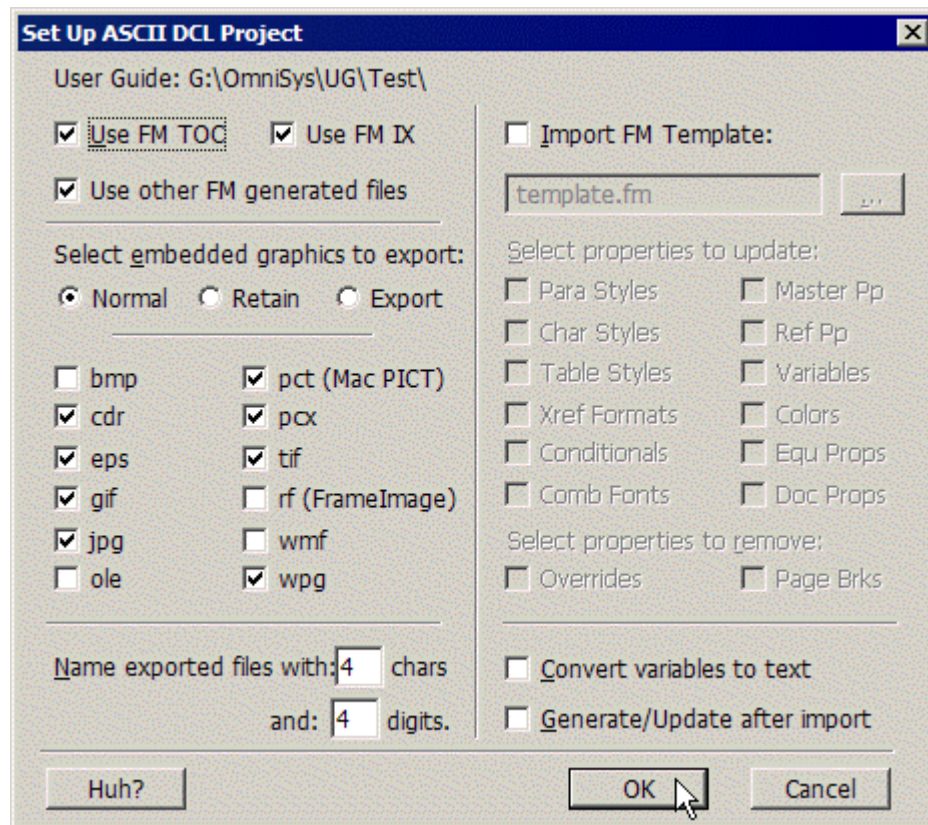
38.3.1 Setting up an ASCII DCL project

When you select ASCII DCL as the output type for a new project, the *Set Up* dialog shown in [Figure 38-2](#) opens. [Table 38-2](#) shows the corresponding settings in the configuration file.

See also:

§3.4 [Choosing project set-up options](#) on page 79

§38.4 [Generating ASCII DCL output](#) on page 1011

Figure 38-2 Set Up ASCII DCL Project

Select graphics to
export

When you specify DCL as the output type, you can take advantage of the fact that **Mif2Go** exports embedded graphics at the same time. You start by choosing some (**Normal**), none (**Retain**), or all (**Export**) graphic formats to export, then select or deselect individual formats:

Normal **Mif2Go** does not export BMP, WMF, or FrameImage (rf) graphics, because these types can be converted successfully for RTF output without creating external files. OLE graphics are not exported either, but **Mif2Go** extracts a WMF graphic from each OLE image; see §31.2.4 [Exporting images and creating files from OLE objects](#) on page 881. You can still select any of these formats to export.

Retain No embedded graphics are exported except those you select individually.

Export All embedded graphics are exported except those you deselect individually.

Names of
exported graphics
files

Mif2Go gives exported graphics files names that start with the first few letters of the name of the FrameMaker file from which they came, followed by an incremental number. You can specify how many letters and how many digits to use for names; the default is four each. See §5.7.4.2 [Naming files produced from embedded graphics](#) on page 134.

Table 38-2 ASCII DCL set-up options and configuration settings

Set-up dialog		Configuration file				
Option		[GraphExport] Setting	Default			Ref.
Select embedded graphics to export	Normal	ImportGraphics=Normal	Normal			31.2.3.5
	Retain	ImportGraphics=Retain	Normal			
	Export	ImportGraphics=Export	Normal			
	Format	Setting for ImportGraphics:	Normal	Retain	Export	31.2.3.5
	bmp	ExportBmpFiles=Yes	No	No	Yes	31.2.3.6
	cdr	ExportCdrFiles=Yes	Yes	No	Yes	
	eps	ExportEpsFiles=Yes	Yes	No	Yes	
	gif	ExportGifFiles=Yes	Yes	No	Yes	
	jpg	ExportJpgFiles=Yes	Yes	No	Yes	
	ole	ExportOleFiles=Yes	No	No	Yes	
	pct	ExportPctFiles=Yes	Yes	No	Yes	
	pcx	ExportPcxFiles=Yes	Yes	No	Yes	
	png	ExportPngFiles=Yes	Yes	No	Yes	
	tif	ExportTifFiles=Yes	Yes	No	Yes	
	rf	ExportRfFiles=Yes	No	No	Yes	
wmf	ExportWmfFiles=Yes	No	No	Yes		
wpg	ExportWpgFiles=Yes	Yes	No	Yes		
Name files with:	chars	ExportNameChars= <i>n</i>	4			5.7.4.2
	digits	ExportNumDigits= <i>n</i>	4			

Note: Selecting `ole` does not create usable external graphics files from OLE objects. This setting is intended primarily for Omni Systems programmers who are debugging **Mif2Go**. The exported file is not useful for any other purpose. See §31.2.4 [Exporting images and creating files from OLE objects](#) on page 881.

38.4 Generating ASCII DCL output

When you specify ASCII DCL as the output type, **Mif2Go** saves all the files in your FrameMaker document in MIF format, and then converts the MIF files to ASCII DCL files (with extension `.dcl`) instead of the usual binary DCL files (with extension `.dcb`), and stops there.

In this section:

§38.4.1 [Including generated files in ASCII DCL output](#) on page 1011

§38.4.2 [Specifying type and file extension for ASCII DCL output](#) on page 1012

§38.4.3 [Exporting embedded graphics via ASCII DCL output](#) on page 1012

38.4.1 Including generated files in ASCII DCL output

You might want generated files to be converted to DCL, or you might not. You can choose individually for Contents and Index files; all others are included or excluded as a group.

```
[Setup]
; UseFrameTOC = Yes (default, exclude for Help formats), or No
UseFrameTOC=No
; UseFrameIX = No (default, exclude for Help formats), or Yes
UseFrameIX=No
; UseFrameGenFiles = Yes (default for all formats) or No
UseFrameGenFiles=No
```

See §5.5 [Converting FrameMaker-generated files](#) on page 124 for more information.

38.4.2 Specifying type and file extension for ASCII DCL output

To produce ASCII DCL output:

```
[Options]
Output = ASCII
```

To specify a file extension:

```
[Setup]
FileSuffix = .dcl
```

38.4.3 Exporting embedded graphics via ASCII DCL output

One of the main reasons for specifying ASCII DCL as the output type is to take advantage of an optional side effect of producing DCL: **Mif2Go** can export embedded graphics at the same time. This method can get embedded graphics out with minimum fuss, and can save time compared to running a full conversion once just to produce graphics.

For example, converting to DCL and exporting graphics takes maybe 10% of the run time for HTML output—but only if you can stick to **Mif2Go** native graphics export, and avoid using FrameMaker export filters (see §5.7.2.2.1 [Understanding when to use FrameMaker export filters](#) on page 129).

To produce graphics first:

```
[Setup]
GraphicsFirst=Yes
```

To export embedded graphics to files:

```
[GraphExport]
ImportGraphics=Export
```

To export embedded graphics only in selected formats:

```
[GraphExport]
ExportXXXFiles=Yes
```

where *xxx* can be any of a long list of graphics formats.

To use **Mif2Go** native graphics export, on the **Mif2Go Export** dialog:

- choose **Write graphics for equations**
- *do not* check **Write only graphics, no text**.

See also:

§3.7.4.2 [Using Mif2Go native graphics processing](#) on page 86

§5.7.3.2 [Processing embedded graphics separately](#) on page 132

§31.2.3 [Exporting and converting embedded graphics](#) on page 877

A WAI marker library for HTML

This section describes example FrameMaker markers to use for Web Access Initiative (WAI) support. The markers themselves are available in file `marklib.fm` in your **Mif2Go** distribution. Topics include:

§A.1 [How to use WAI markers](#) on page 1013

§A.2 [Table markers](#) on page 1013

§A.3 [Graphic markers](#) on page 1014

§A.4 [Link markers](#) on page 1016

A.1 How to use WAI markers

Marker text is limited to 256 characters. If you need more, you can insert another marker of the same type in the same paragraph, and continue the text; **Mif2Go** concatenates the text in successive markers of the same type. Empty markers are ignored by **Mif2Go**.

If you are using the FrameMaker version of this material (`marklib.fm`), make sure you have **View > Text Symbols** checked so you can see the marker symbols. You can copy and paste these markers into your own documents as needed; when you paste, the Marker Type is automatically added to your document's list of available types.

A.2 Table markers

[Table A-1](#) lists the special marker types you can use to apply WAI attributes to tables. The **T** column in contains an example of each table marker. The “Table A-2” columns show where in [Table A-2](#) on page 1014 you can find another example of the same marker, located in a cell appropriate for the marker purpose.

Table A-1 Special marker types for WAI table attributes

T	marker type	Purpose	HTML attribute	Valid content	Table A-2	
					Col	Row
—	CellAbbr	Abbreviation for header-cell text	abbr	Any text	2	6
—	CellAxis	Cell's data category	axis	Any text	5	3
—	CellGroup	Header cell's ColGroup or RowGroup property *	id/headers or scope	col row	4,7 1	1 4,7,11
—	CellScope	Header cell's scope	scope	col colgroup row rowgroup	1 4,7 3 1	3 1 4 4,7,11
—	CellSpan	Assign id="span" to a header cell	id/headers	Non-empty	5,8 3	1 4,10
—	TableSummary	Text describing the table	summary	Any text	Upper left corner	
—	TableTitle	Text shown as a mouseover	title	Any text	Upper left corner	

* See §26.2.1 [Specifying group properties for header cells](#) on page 766 for more information.

For more information about table marker types, see:

§25.4.3.3 Using a custom marker for table summary or title on page 762

§26.2.4 Assigning table-cell attribute values with custom markers on page 772

Each of the paragraph formats used in Table A-2 has a name that represents the role it plays in the table: *Body Cell*, *Col Group*, *Col Head*, *Row Group*, *Row Head*, and *Table Footer*. Columns are numbered in row 3; rows are numbered in column 3. The content of other column and row header cells indicates the kind of WAI markup provided by the markers they contain. The *Col Group* and *Row Group* header cells each contain two markers: a **CellScope** marker at the beginning of the text, and a **CellGroup** marker at the end. Body cells are all numbered by column and row. This paragraph ends with the table anchor for Table A-2:

Table A-2 Examples of WAI table markers

This corner cell contains a TableSummary marker here, and TableTitle marker here.			Col Group 1		Col Span 1		Col Group 2		Col Span 2	
Col 1	Col 2	Col 3	Col 4	Col 5 with Axis	Col 6	Col 7	Col 8	Col 9		
Row Group 1	Row Span 3	Row 4	C4R4	C5R4	C6R4	C7R4	C8R4	C9R4		
		Row 5	C4R5	C5R5	C6R5	C7R5	C8R5	C9R5		
	Row Head 6 with Abbr	Row 6	C4R6	C5R6	C6R6	C7R6	C8R6	C9R6		
Row Group 2	Row Head 7	Row 7	C4R7	C5R7	C6R7	C7R7	C8R7	C9R7		
	Row Head 8	Row 8	C4R8	C5R8	C6R8	C7R8	C8R8	C9R8		
	Row Span 4	Row 9	C4R9	C5R9	C6R9	C7R9	C8R9	C9R9		
		Row 10	C4R10	C5R10	C6R10	C7R10	C8R10	C9R10		
Row Group 3	Row Head 11	Row 11	C4R11	C5R11	C6R11	C7R11	C8R11	C9R11		
	Row Head 12	Row 12	C4R12	C5R12	C6R12	C7R12	C8R12	C9R12		
	Row Head 13	Row 13	C4R13	C5R13	C6R13	C7R13	C8R13	C9R13		
Footer			FC4	FC5	FC6	FC7	FC8	FC9		

A.3 Graphic markers

Table A-3 lists the marker types you can use for graphics. The **I** column contains an example of each graphic marker.

Table A-3 Special marker types for WAI graphic attributes

I	marker type	Purpose	HTML attribute	Valid content
—	GraphAlt	Brief text description of the graphic	<code>alt</code>	Any text
—	GraphLongdesc	Link to a file or Web page containing more information about the graphic	<code>longdesc</code>	File name or URL
—	GraphTitle	Text description of the graphic; use for browsers that do not support the <code>longdesc</code> attribute	<code>title</code>	Any text

GraphAlt is for a name or brief text description of the graphic, to identify the image for the visually impaired. Some browsers might display the `alt` text as a “tool tip”.

GraphLongdesc is for the name of a file, or URL of a Web page, that provides a longer description of the graphic than is afforded by **GraphAlt**.

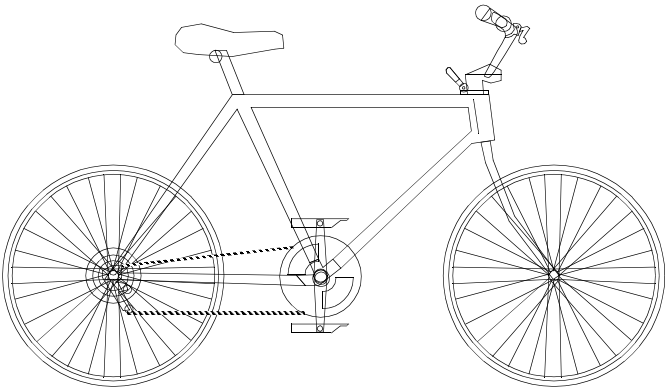
GraphTitle is also for a longer description of the graphic; use this marker as an alternate to (or in addition to) the **GraphLongdesc** marker, for browsers that do not support the `longdesc` attribute. Some browsers might display the `title` text as a “tool tip”.

You should experiment to see how each of these attributes is rendered by the browsers you intend people to use.

You cannot insert a marker in a graphic itself, unless you include a text frame also. Place the markers anywhere before a graphic’s anchor; they will apply to the next graphic in the flow. Once used, the markers do not persist to affect any following graphics; in fact, the nature of these markers is such that they should never repeat exactly for two different graphics.

The graphic anchor for [Figure A-1](#) is in a paragraph that follows the figure title. The figure title itself contains examples of the three **Graph*** markers, all inserted in the word “travel”.

Figure A-1 Primary travel method in near future



A.4 Link markers

Table A-4 lists the marker types you can use for links. The **I** column contains an example of each link marker.

Table A-4 Special marker types for WAI link attributes

I	marker type	Purpose	HTML attribute	Valid content
—	LinkClass	Set link display properties in CSS	<code>class</code>	CSS class name
—	LinkTitle	Descriptive title for link destination	<code>title</code>	Any text

If you give a link the `title` attribute, the content of the attribute is displayed when mousing over the link, instead of the content of the `href` attribute. A **LinkTitle** marker is here, and it applies to the link to §A [WAI marker library for HTML](#) on page 1013.

B Distribution files

The files listed in [Table B-1](#) are included in each **Mif2Go** distribution. *Read instructions before installing*; see §1.3.3 [Install Mif2Go](#) on page 56.

Table B-1 Mif2Go distribution files

Category	Where installed ¹	File name	Description
Executable	\common\bin	dcl.exe	Document Coding Language (DCL) program
		exwmf.exe	Embedded-graphics extractor
		pprtf.exe	RTF pretty printer
		runfm.exe	Command-line application for unattended runs
		setini.exe	Utility for changing configuration settings
DCL library	\common\bin	dcx.dll	Converter between ASCII and binary DCL
		drxml.dll	DCL reader for MIF files
		dwhtm.dll	DCL writer for HTML/XML files
		dwinf.dll	DCL writer for .inf (document information) files
		dwrftf.dll	DCL writer for RTF files
		msvcrt40.dll	Microsoft Windows C++ run-time library
Plug-in files	FrameMaker plug-in directory	m2rbook.dll	Mif2Go FrameMaker plug-in interface
		m2gframe.dll	Library for m2rbook.dll
Configuration	\common\system\config	omsys.ini	Base configuration template
	\common\local\config	local_omsys.ini	Editable base configuration file
	\m2g\system\config	m2*_config.ini	Output-type configuration templates
	\m2g\local\config	local_m2*_config.ini	Editable output-type configuration files
Macros	\m2g\system\macros	m2*_macros.ini	Macro configuration templates
	\m2g\local\macros	local_m2*_macros.ini	Editable macro configuration files
Documentation	\m2g\usersguide	ugmif2go.chm	Mif2Go User's Guide in HTML Help format
OmniHelp	\common\omnihelp	ohvhtmNW.zip	OmniHelp control files for HTML 4.01
		ohvxmlNW.zip	OmniHelp control files for XHTML 1.0
		ohvm2gNW.zip	OmniHelp customization files

¹ For complete installation instructions, see §1.3.3 [Install Mif2Go](#) on page 56.

C Document and conversion files

This section describes the files **Mif2Go** creates for and during a conversion project, and shows where to find these files and what to do with them. Topics include:

- §C.1 [Locating document and conversion files](#) on page 1019
- §C.2 [Identifying conversion files](#) on page 1020
- §C.3 [Renaming or relocating the Mif2Go project file](#) on page 1026
- §C.4 [Renaming or relocating the Mif2Go FileID file](#) on page 1027
- §C.5 [Working with reference files for HTML or XML](#) on page 1027

C.1 Locating document and conversion files

When you use **Mif2Go** to convert a FrameMaker document, you specify a project directory for converted and generated document files (see §2.6 [Establishing a conversion environment](#) on page 74). By default, **Mif2Go** places output files in the *input directory*: the directory where your FrameMaker document is located. If you accept the default (*not recommended*), the project directory for your conversion is the same as the input directory. It is better practice to create a different directory for output files. [Table C-1](#) shows what document files are in which directory.

Table C-1 Location of document files

Directory	File name	Description
Input	<i>MyDoc.book</i> (& *.fm) or <i>MyDoc.fm</i>	Your original FrameMaker document
Output	<i>MyDoc.mif</i> (& *.mif)	Your FrameMaker document in MIF format (unless you instruct Mif2Go to delete MIF files)
	*.rtf or *.htm, etc.	Converted and generated document file(s), in the output format you specify

Mif2Go also creates several conversion files, and places some in the input directory, some in the project directory. [Table C-2](#) lists the basic conversion files in each directory. For more information see §C.2 [Identifying conversion files](#) on page 1020.

Table C-2 Location of conversion files

Directory	File name	Description
Input	<i>MyDoc.prj</i>	Project information Mif2Go creates for your FrameMaker document
	<i>mif2go.ini</i>	List of FileIDs assigned to the files Mif2Go generates from your FrameMaker document (not produced if you select FrameMIF as the output type)
Output	<i>_m2g_log.txt</i>	Conversion event log (optional)
	<i>_m2output.ini</i>	Configuration file containing settings to produce the output file(s); put there by Mif2Go if you convert via FrameMaker, or by you (or a script you provide) if you convert via command line
	*.grx	Graphics information entries for each file in your document that contains graphics which you direct Mif2Go to export via FrameMaker filters—even if you check Do not write graphics on the <i>Export</i> dialog
	*.ref	Cross-reference entries for each file in your document (HTML output only)

You might find other conversion files in the project directory as well, depending on the type of output you specify and the conversion options you select. See §C.2.3 [Additional conversion files](#) on page 1021.

C.2 Identifying conversion files

Mif2Go tends to litter your input and output directories with conversion files. It is a good idea to know what these files are for, and what you should and should not do with them.

In this section:

§C.2.1 [Conversion files created during set-up](#) on page 1020

§C.2.2 [Files created during conversion](#) on page 1021

§C.2.3 [Additional conversion files](#) on page 1021

§C.2.4 [What not to do with conversion files](#) on page 1025

C.2.1 Conversion files created during set-up

The first time you set up a conversion project for a particular FrameMaker document, **Mif2Go** creates for that document three conversion files:

- a document-specific *project file*
- a document-specific *FileID file*
- an output-specific starting project *configuration file*:

<u>Conversion file</u>	<u>Default file name</u>	<u>File contents</u>
Project	<i>MyDoc.prj</i>	Information Mif2Go collects about conversion-project names, types, paths, and states.
FileID	<i>mif2go.ini</i>	Output-file identifiers called FileIDs, mapped to the names of your FrameMaker files; Mif2Go assigns these identifiers.
Configuration	<i>_m2output.ini</i>	Settings and options that dictate how your document will be converted.
	<i>_m2output_html.ini</i> or	Settings that are specific to this document.
	<i>_m2output_word.ini</i>	

Mif2Go places the project file and the FileID file in the same directory as your FrameMaker document, and places the starting project configuration file in the project directory.

Before Mif2Go set-up:

Input directory Project directory

MyDoc.book
Chapter1.fm
 ...
ChapterN.fm

After Mif2Go set-up:

Input directory Project directory

MyDoc.book *_m2output.ini*
Chapter1.fm
 ...
ChapterN.fm
MyDoc.prj
mif2go.ini

*Project file and
FileID file*

If you run **Mif2Go** from within FrameMaker, the project and FileID files are created when you click **Set Up Mif2Go Export...** on the FrameMaker **File** menu:

<i>MyDoc.prj</i>	Project file for conversions from <i>MyDoc.book</i> (or <i>MyDoc.fm</i>)
<i>mif2go.ini</i>	FileID file for conversions from <i>MyDoc.book</i> (or <i>MyDoc.fm</i>)

See §C.3 [Renaming or relocating the Mif2Go project file](#) on page 1026 for more information about these two conversion files.

Configuration file **Mif2Go** places the starting project configuration file in the project directory. If you run **Mif2Go** from within FrameMaker, the project directory is the location you specify via the Choose Project **Browse** button (...). The configuration file is one of those listed in [Table 30-5](#) on page 859. **Mif2Go** also places a document-specific configuration file in subdirectory `_config`, parallel to the project directory. This file contains settings that apply only to the particular FrameMaker document you are converting. See §30.3 [Including document-specific configuration files](#) on page 852.

C.2.2 Files created during conversion

After the first time you run a conversion, in addition to the files described in §C.2.1 [Conversion files created during set-up](#) on page 1020, you might find several other conversion files created by **Mif2Go**: graphics information files and reference files, for example.

Graphics information files If your document includes graphics, and **Mif2Go** converts them using FrameMaker export filters (see §5.7.2.2 [Using FrameMaker graphic export filters](#) on page 129), the project directory will contain a *graphics information file* for each FrameMaker file in your document. A graphics information file has the same name as its FrameMaker file, but with extension `.grx`.

Before document conversion:

Input directory Project directory

`MyDoc.book` `_m2output.ini`
`Chapter1.fm`
 ...

`ChapterN.fm`
`MyDoc.prj`
`mif2go.ini`

After document conversion:

Input directory Project directory

`MyDoc.book` `_m2output.ini`
`Chapter1.fm` `MyDoc.mif`
 ... `MyDoc.rtf or *.html or`
 `*.xml or *.dita`

`ChapterN.fm` `Chapter1.grx`
`MyDoc.prj` ...
`mif2go.ini` `ChapterN.grx`

Reference files If you are converting to HTML or XML, you will also find a *reference file* corresponding to each FrameMaker file, with extension `.ref`; see §C.5 [Working with reference files for HTML or XML](#) on page 1027.

C.2.3 Additional conversion files

After you run **Mif2Go** additional conversion files might be present in the project directory, depending on the following:

- Which kind of output type you specify; see:
 - §C.2.3.1 [RTF conversion files](#) on page 1022
 - §C.2.3.2 [HTML/XML conversion files](#) on page 1022
 - §C.2.3.3 [MIF or DCL conversion files](#) on page 1025
- Whether you are generating on-line help; see:
 - §C.2.3.1.2 [WinHelp files](#) on page 1022
 - §C.2.3.2.4 [MS HTML Help files](#) on page 1024
 - §C.2.3.2.5 [OmniHelp files](#) on page 1024
 - §C.2.3.2.6 [JavaHelp or Oracle Help files](#) on page 1024
- What options you specify for converting graphics; see:
 - §3.6 [Converting documents](#) on page 82

- §5.7 [Processing graphics](#) on page 126
- §6.14 [Managing graphics for print RTF](#) on page 186
- §8.6 [Managing graphics for WinHelp](#) on page 263
- §23 [Including graphics in HTML](#) on page 703
- §31 [Working with graphics](#) on page 869

C.2.3.1 RTF conversion files

Conversion files **Mif2Go** places in the project directory depend on which RTF output type you specify:

- §C.2.3.1.1 [Print RTF \(Word or WordPerfect\) files](#) on page 1022
- §C.2.3.1.2 [WinHelp files](#) on page 1022

C.2.3.1.1 Print RTF (Word or WordPerfect) files

If you are converting to a print RTF format (Word or WordPerfect), **Mif2Go** creates no additional conversion files that are specific to RTF. If you use default conversion settings, you should see the following files in the project directory:

<code>*.grx</code>	Graphics information entries for each file in your document that contains graphics exported via FrameMaker filters.
<code>*.mif</code>	Your document files in MIF format
<code>_m2rtf.ini</code>	Configuration file
<code>MyDoc.rtf</code>	Your document in RTF format

C.2.3.1.2 WinHelp files

If you are generating WinHelp, all graphics in anchored frames in your FrameMaker document will appear as individual WMF graphics files. If you use default conversion settings, you should see the following files in the project directory, in addition to those listed in §C.2.3.1.1 [Print RTF \(Word or WordPerfect\) files](#) on page 1022:

<code>*.bct</code>	Contents entries for each file in your document
<code>*.wmf</code>	Graphics
<code>MyDoc.cnt</code>	WinHelp contents
<code>MyDoc.hpj</code>	WinHelp project

C.2.3.2 HTML/XML conversion files

Conversion files **Mif2Go** places in the project directory depend on which output type you specified:

- §C.2.3.2.1 [Standard HTML or XML files](#) on page 1023
- §C.2.3.2.2 [DITA XML files](#) on page 1023
- §C.2.3.2.3 [DocBook XML files](#) on page 1023
- §C.2.3.2.4 [MS HTML Help files](#) on page 1024
- §C.2.3.2.5 [OmniHelp files](#) on page 1024
- §C.2.3.2.6 [JavaHelp or Oracle Help files](#) on page 1024
- §C.2.3.2.7 [Eclipse Help files](#) on page 1025

C.2.3.2.1 Standard HTML or XML files

Unless you specify otherwise, all graphics in anchored frames in your FrameMaker document will appear as individual JPEG graphics files. If you use default conversion settings, you should see the following files in the project directory:

<code>*.grx</code>	Graphics information entries for each file in your document that contains graphics exported via FrameMaker filters.
<code>*.jpg</code>	Exported graphics
<code>*.htm</code>	Your document in HTML format, if the output type is HTML, XHTML, or any HTML-based Help type
<code>mydoc.lst</code>	List of FrameMaker files being processed
<code>*.mif</code>	Your document in MIF format
<code>*.ref</code>	Cross-reference entries for each file in your document.
<code>*.xml</code>	Your document in XML format, if the output type is XML
<code>lp.gif</code>	Spacer graphic
<code>local.css</code>	Cascading Style Sheet
<code>_m2*ml.ini</code>	Starting project configuration file

C.2.3.2.2 DITA XML files

With default conversion settings you should see the following files in the project directory:

<code>*.dita</code>	Your document in DITA XML format
<code>*.ditamap</code>	DITA map for the book file and for each chapter file
<code>*.dtf</code>	ASCII DCL information needed to build the book map, for each chapter in your document
<code>*.grx</code>	Graphics information entries for each file in your document that contains graphics exported via FrameMaker filters
<code>*.jpg</code>	Exported graphics
<code>*.mif</code>	Your document in MIF format
<code>*.ref</code>	Cross-reference entries for each file in your document.
<code>mydoc.lst</code>	List of FrameMaker files being processed
<code>local.css</code>	Cascading Style Sheet
<code>_m2dita.ini</code>	Configuration file

C.2.3.2.3 DocBook XML files

With default conversion settings you should see the following files in the project directory:

<code>*.ent</code>	Your chapter files in DocBook XML format
<code>*.xml</code>	Your book file in DocBook XML format
<code>*.grx</code>	Graphics information entries for each file in your document that contains graphics exported via FrameMaker filters
<code>*.jpg</code>	Exported graphics
<code>*.mif</code>	Your document in MIF format
<code>*.ref</code>	Cross-reference entries for each file in your document.

<i>mydoc.lst</i>	List of FrameMaker files being processed
<i>local.css</i>	Cascading Style Sheet
<i>_m2docbook.ini</i>	Configuration file

C.2.3.2.4 MS HTML Help files

In addition to the files listed in §C.2.3.2.1 [Standard HTML or XML files](#) on page 1023, with default conversion settings you should see the following files in the project directory:

*.bhc	Contents entries
*.bhk	Index entries
<i>MyProj.hha</i>	Index headers
<i>MyProj.hhp</i>	HTML Help project
<i>MyProj.hhc</i>	Contents in HTML format
<i>MyProj.hhk</i>	Index in HTML format
<i>MyProj.hht</i>	List of CSH IDs for HTML Help
<i>MyProj.lst</i>	List of FrameMaker files being processed
<i>MyProjIX.bha</i>	Index entries from your document's index

C.2.3.2.5 OmniHelp files

In addition to the files listed in §C.2.3.2.1 [Standard HTML or XML files](#) on page 1023, with default conversion settings you should see the following files in the project directory:

*.bha	Aliases, such as the IDH_ identifiers, from newlinks
*.bhc	Contents entries
*.bhk	Index entries
*.bhl	Related items (ALinks)
*.bhs	Search items
<i>MyDoc.lst</i>	List of FrameMaker files being processed
<i>MyDoc.oha</i>	Context-sensitive help entries
<i>MyDoc.ohc</i>	Contents entries
<i>MyDoc.ohk</i>	Index entries
<i>MyDoc.ohl</i>	Related-topics entries
<i>MyDoc.ohs</i>	Full-text search entries
<i>MyDoc.ohx</i>	Settings from project configuration file <i>_m2omnihelp.ini</i>
<i>MyDocohp.htm</i>	Frameset to load in browser

C.2.3.2.6 JavaHelp or Oracle Help files

In addition to the files listed in §C.2.3.2.1 [Standard HTML or XML files](#) on page 1023, with default conversion settings you should see the following files in the project directory:

*.bhc	Contents entries
*.bhk	Index entries
*.bhm	Index headers
<i>MyProj.hs</i>	Helpset file

<i>MyProj.jhm</i>	Map file
<i>MyProj.lst</i>	List of files being processed
<i>MyProjIX.bha</i>	Index entries from your document's index
<i>MyProjIndex.xml</i>	Index entries from your document's index, in XML format
<i>MyProjTOC.xml</i>	Contents entries from your document's TOC, in XML format

C.2.3.2.7 Eclipse Help files

In addition to the files listed in §C.2.3.2.1 [Standard HTML or XML files](#) on page 1023, with default conversion settings you should see the following files in the project directory:

<i>*.bhc</i>	Contents entries
<i>MyDoc.lst</i>	List of FrameMaker files being processed
<i>plugin.xml</i>	Manifest file
<i>toc.xml</i>	Contents file

C.2.3.3 MIF or DCL conversion files

Conversion files **Mif2Go** places in the project directory depend on which output type you specified:

§C.2.3.3.1 [FrameMaker MIF files](#) on page 1025

§C.2.3.3.2 [ASCII DCL files](#) on page 1025

C.2.3.3.1 FrameMaker MIF files

When you choose **Frame MIF only**, **Mif2Go** creates no additional conversion files; and in fact does not produce *mif2go.ini*. If you use default conversion settings, you should see the following files in the project directory:

<i>*.mif</i>	Your document files in MIF format
<i>_m2mif.ini</i>	Configuration file

C.2.3.3.2 ASCII DCL files

When you choose **ASCII DCL only**, **Mif2Go** creates no additional conversion files. If you use default conversion settings, you should see the following files in the project directory:

<i>*.dcl</i>	Your document in ASCII DCL format
<i>*.mif</i>	Your document files in MIF format
<i>_m2dcl.ini</i>	Configuration file

C.2.4 What not to do with conversion files

Mostly you should leave conversion files alone. But if you must:

[Move files with care](#)

[Delete files with care](#)

[Modify files with care.](#)

Move files with care

Leave conversion files in place unless you are very sure you will not need to run all or part of the conversion again.

For example, if you move document, project, and FileID files to another directory, **Mif2Go** still expects to find the FileID file in the original location. This is because the

configuration file contains an absolute (full-path) reference to the location of the FileID file at the time the project was created. *If Mif2Go does not find a FileID file in the referenced directory, Mif2Go creates a new one there.* To get around this problem, edit the configuration file to change the reference; see §C.4 [Renaming or relocating the Mif2Go FileID file](#) on page 1027.

If you do move your entire project to another directory, delete all *.ref files for that project, and also delete all *.ref files for any project that links to the one you are moving.

Delete files with care

You may delete reference (.ref) files as a precursor to a full rebuild. Deleting reference files created in a prior conversion can make the next conversion of the same document run faster. This is safe practice only if both of the following apply:

- You are about to reconvert an entire book.
- No files *outside* the book reference any files *in* the book.

Otherwise, you might get broken cross references from other files, until you reconvert those files also. If you reconvert only one chapter after deleting .ref files, any cross references from that chapter to other chapters (or HTML split files) will be wrong until those other chapters have been reconverted also. See §C.5 [Working with reference files for HTML or XML](#) on page 1027. See also §35.4.3 [Understanding when not to delete .ref and .htm files](#) on page 959

Modify files with care

If you add, rename, or remove any files in your FrameMaker book between conversions, you might need to edit certain conversion files. See §C.4 [Renaming or relocating the Mif2Go FileID file](#) on page 1027 and §C.5 [Working with reference files for HTML or XML](#) on page 1027. To modify a conversion file, use a text editor such as Notepad. Otherwise, do not change the content of any conversion files except for the configuration file, unless you know what you are doing.

C.3 Renaming or relocating the Mif2Go project file

By default, **Mif2Go** creates a project file with the same base name as your FrameMaker document, and extension .prj. By default the project file is located in the same directory as your FrameMaker document files. However, in a configuration file you can specify a different name or location for the project file:

```
[Setup]
; PrjFileName = name of project file that references this directory
PrjFileName=D:/path/to/yourbook.prj
```

If **Mif2Go** cannot find the specified project file, **Mif2Go** looks for a file named mif2go.prj.

Path to and paths within the project file must be absolute

You must specify an absolute path to the project file. Also, the path values **Mif2Go** creates within the project file must be absolute. Some users always keep book and chapter files in the same directory; others do not. Sometimes all books are in one place, all chapters in a few others. It is possible for the book and every chapter in it to be in a different place. Yet you need a single project directory for the project. An absolute path makes the conversion location predictable and stable.

Copied configurations are automatically updated

A configuration file created by **Mif2Go** specifies the default name and the input directory for PrjFileName. If you copy a configuration file from another location to the project directory, then set up a new conversion, **Mif2Go** updates the copied configuration file to fix the path to the new project file.

C.4 Renaming or relocating the Mif2Go FileID file

By default, for a new project **Mif2Go** creates a FileID file named `mif2go.ini` in the same directory as your FrameMaker document files. However, in the starting project configuration file you can specify a different name or location for the FileID file:

```
[Setup]
; IDFileName = name of file that contains [FileIDs] for this project
IDFileName=D:/absolute/path/to/mif2go.ini
```

*Copied
configurations are
not updated*

A configuration file created by **Mif2Go** specifies the default name and the input directory for `IDFileName`. If you copy a configuration file from another location to the project directory, then set up a new conversion, **Mif2Go** does *not* fix the path to the FileID file, because you could be using a centralized version of the latter; **Mif2Go** cannot tell. If you are not using a centralized version, you might have to edit the path to the FileID file.

If you have multiple projects that use the same FrameMaker document files, but different project directories, probably you should edit all the configuration files so they all point to the same copy of `mif2go.ini`; you can put that copy anywhere that is convenient. Then, for example, all your projects would use the same FileID for graphics produced from the same FrameMaker file.

*FileIDs in old
configuration files*

If you have just upgraded **Mif2Go** from a prior version that kept FileID information in the main configuration file, and you are still using that configuration file, you can override the list in the FileID file with the list in your configuration file; see §5.3.4.4 [Keeping legacy FileIDs in the configuration file](#) on page 122.

C.5 Working with reference files for HTML or XML

If you are converting to HTML or XML, for each FrameMaker file in your document **Mif2Go** creates a *reference file* of information about both ends of every cross reference and every link from, to, or within that FrameMaker file. **Mif2Go** uses this information to figure out and correct HTML links. *Do not edit **Mif2Go** reference files.*

A reference file has the same name as its FrameMaker file, but with extension `.ref`. Reference files are placed in the project directory, along with HTML or XML output files. However, you do not need to distribute reference files with your output. They are for **Mif2Go** internal use only.

In this section:

§C.5.1 [Understanding how reference files work](#) on page 1027

§C.5.2 [Resolving forward references with a second pass](#) on page 1028

§C.5.3 [Using reference files to enable links to other documents](#) on page 1028

C.5.1 Understanding how reference files work

When **Mif2Go** encounters in one FrameMaker file a cross reference or link to another file in the document, **Mif2Go** looks for the reference file produced when that other file was processed—even if that other FrameMaker file is in a different directory. If the reference file is present, **Mif2Go** uses the information it contains to create a link to the correct split part of that other file.

When **Mif2Go** encounters, in the reference file for the current FrameMaker file, a marker for a cross reference from another file, **Mif2Go** updates the other file to point to the correct split part of the current file. The net effect is that after **Mif2Go** processes the last file in a document, all links between all files are correct.

During conversion, **Mif2Go** updates HTML files that are already in the project directory if they contain links to FrameMaker files that **Mif2Go** has not yet processed; and if those FrameMaker files are then split, the links in the HTML output files are updated to point to the correct split parts. This means you must *leave all HTML files in the project directory* until the entire conversion is finished.

Reference files are fully regenerated only when you convert the entire document they deal with, along with any other files that reference, or are referenced by, files in that document. Until you complete the conversion, some of the links to files that are split will be wrong. The HTML output files might also be updated as part of the process, and must be left where created for this to work.

If you find an entry followed by a double asterisk in a reference file; for example:

```
[Links]
ca871052=49CFR191.htm#Xbw1006201 **
```

The double asterisk means that **Mif2Go** was unable to resolve the reference. See §D.2.7 [Check for broken links \(HTML or XML output\)](#) on page 1033.

C.5.2 Resolving forward references with a second pass

Forward links to FrameMaker files that have not yet been processed might not be resolved the first time you run a conversion. This is a problem when macros are involved, but only for cross references to other FrameMaker files (as opposed to hypertext links or URLs, which do not have this issue). The problem can also be caused by latency issues in Windows shell operations.

Mif2Go does not know the final reference while processing for the first time a macro that includes a forward cross reference, because the destination output file does not yet exist. When **Mif2Go** processes the destination file, **Mif2Go** goes back to fix up the cross reference; but this does not work if a macro is involved, because in that case **Mif2Go** does not know exactly where in the source file the unfixed reference is located.

However, if there is an accurate `.ref` file when **Mif2Go** processes the original reference, the destination is known, and so no fix-up is needed.

The solution is to run the conversion twice. The first run populates the `.ref` file correctly, and the second run uses that information to fix up forward references. Of course *you must not delete the `.ref` file in between the two runs*.

C.5.3 Using reference files to enable links to other documents

Mif2Go does not delete reference files after conversion, because they could be needed at a later time if other FrameMaker documents contain references to or from any previously converted document.

If the FrameMaker document you are converting contains links to other FrameMaker documents that were already converted, see §19.6.4 [Enabling links to files in other projects](#) on page 623 for additional settings.

See also:

§35.4 [Clearing out old files before converting](#) on page 957

D Technical support for Mif2Go

Omni Systems can provide effective technical support for **Mif2Go** when you provide complete, concise information. We always do our best to help; when you do your part, we can do our part more quickly and effectively. Topics include:

§D.1 [Evaluation version is different](#) on page 1029

§D.2 [Things to check first](#) on page 1029

§D.3 [How to request help](#) on page 1035

Zip your files! *Do not send unzipped FrameMaker files to Omni Systems.
Do not send files larger than 1 MB.*

D.1 Evaluation version is different

If you are using the evaluation version of **Mif2Go**, and the problem you see is that some of the text in your output has been replaced by nonsense, *this is intentional*:

[Evaluation version alters text](#)

[Get the latest release](#)

[Get a license](#)

[Delete evaluation components if necessary](#)

*Evaluation
version alters text*

The evaluation version of **Mif2Go** replaces some of your text with excerpts from Lewis Carroll's poem "Jabberwocky". This ensures that you can evaluate **Mif2Go** without restrictions on the size or complexity of your project, yet you cannot produce commercially usable output until you actually purchase a license. For additional information about using the evaluation version, see `readme.txt` in `%OMSYSHOME%\m2g\zip`.

*Get the latest
release*

If you encounter some other problem with the evaluation version, first download the newest `sampXXuYY.zip` file, and reinstall; then try the conversion again. To obtain the latest release of the evaluation version, download file `sampXXuYY.zip` from the following location:

<http://mif2go.com>

Get a license

To purchase a license for **Mif2Go**, see:

<http://mif2go.com>

*Delete evaluation
components if
necessary*

Once you have installed a licensed copy of **Mif2Go**, you should no longer see any of your text replaced by excerpts from Jabberwocky. However, if you are still seeing these excerpts, some components of the evaluation version are still present on your system; and one of them, `drmfif.dll`, is located in a directory that is earlier on your system `PATH` than the licensed version. The evaluation version of this file is larger than the production version. Use Windows Explorer to search your hard disk for `drmfif.dll`, and delete any copies of this file that are 320K in size rather than 317K.

D.2 Things to check first

Before you holler for help, try the following:

§D.2.1 [Examine your conversion log file](#) on page 1030

§D.2.2 [Check your Mif2Go installation](#) on page 1030

§D.2.3 [Check for missing Microsoft components](#) on page 1032

- §D.2.4 [Check the Mif2Go User's Guide](#) on page 1032
- §D.2.5 [Check path names, file names, and drive location](#) on page 1032
- §D.2.6 [Check for file corruption](#) on page 1032
- §D.2.7 [Check for broken links \(HTML or XML output\)](#) on page 1033
- §D.2.8 [Restart FrameMaker, reboot system](#) on page 1033
- §D.2.9 [Check your version of Mif2Go](#) on page 1034
- §D.2.10 [Narrow down the problem](#) on page 1035

D.2.1 Examine your conversion log file

By default, **Mif2Go** writes conversion errors and warnings to a log file in your project directory. If the information in the log file does not reveal the cause of the problem, try changing the log options to capture more information. See §5.2 [Logging conversion events](#) on page 115.

No log errors or warnings? Next: [Check your Mif2Go installation](#)

D.2.2 Check your Mif2Go installation

Sometimes supporting files are not where they need to be. See if you are getting one of these error messages:

[DCL NT console driver has stopped working](#)
[Error processing m2rbook...](#)
[Could not run DCL filter...](#)
[Arguments unacceptable](#)
[OmniHelp Loading...](#)

*DCL NT console
driver has
stopped working*

Check whether:

- %OMSYSHOME% is defined as a System environment variable
- %OMSYSHOME%\common\bin is on your System (not User) execution PATH.

See §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54.

*Error processing
m2rbook...*

If you get an error message such as the following when you run **Mif2Go** from the FrameMaker File menu:

Error Processing m2rbook command

Generally this indicates one of the following problems:

- A corrupted **Mif2Go** project (.prj) file; see §C.1 [Locating document and conversion files](#) on page 1019. You can try deleting your existing .prj file, and then restarting FrameMaker. However, if your FrameMaker book file is on a network server, you might need to work with a local copy. Update it from the network copy when you start, and copy back changes when you are finished, assuming others also need to work on the same book.
- A missing document file, if you get the error while **Mif2Go** is checking links.
- A problem with the file system where your FrameMaker book is located; this happens mostly with network server file systems; it seems to be an intermittent problem of FrameMaker not getting write access to the directory that contains the book file. This is more a FrameMaker problem than a **Mif2Go** problem, and might also manifest as trouble saving files, or trouble with graphics.

Other, similar error messages include the following:

Error initializing m2rbook.
 Error processing m2rbook notification.


```
Error processing m2rbook hypertext command.
Error processing m2rbook dialog event.
```

Each message is issued under a different condition, usually as a result of an internal problem in FrameMaker, but sometimes as a result of a **Mif2Go** problem. If you can get the error message to display consistently, send a test case, prepared as described in §D.3 [How to request help](#) on page 1035.

*Could not run
DCL filter...*

If the FrameMaker Console reports an error such as the following:

```
Mif2go failed for file: G:\HTML\First.htm
type 1, code 2, at: Thu Jul 25 10:07:37 2012
Could not run DCL filter or other program.
File not found.
```

This is an installation problem: **Mif2Go** could not find one or more executable files. This error usually means that the PATH environment variable is not set to include %OMSYSHOME%\common\bin, where the DLLs **Mif2Go** is looking for reside. You must set the SYSTEM PATH, not the USER PATH; see §1.3.3 [Install Mif2Go](#) on page 56. Another possibility is that %OMSYSHOME% is not set as a SYSTEM (not USER) environment variable, or is not set to the path where you actually installed the executables.

Note: After you make environment settings, you must reboot Windows before they take effect.

Make sure your zip program did not put an extra directory name in the path when you unpacked the distribution archive. For example, if %OMSYSHOME% is defined as C:\omsys, make sure the executables are in C:\omsys\common\bin, and not in C:\omsys\m2g_full_54\common\bin.

If files are all present in the right places, look at them in Windows Explorer. Right-click each DLL, choose **Properties**, and look at the **Security** tab. Make sure that everyone has permission to execute, preferably “full control”. Windows “security” often disables whatever came from a downloaded .zip file by limiting permissions for any executables.

*Arguments
unacceptable*

If the FrameMaker Console reports an error such as the following:

```
Mif2go failed for file: G:\HTML\Test.htm
code 1 at: Wed Aug 17 15:05:57 2012
Arguments unacceptable.
```

This usually means that one or more required DLL files are not installed where Windows can find it; see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 54. Sometimes this error occurs when an administrator “cleans up” system directories and removes an essential file (such as msvcrt40.dll, the Microsoft C run-time library for Visual C++ 4.x). Make sure all DLLs are in the required places.

You can get an “Arguments unacceptable” error message if compilation fails when you are generating WinHelp or HTML Help, and you set CompileHelp=Yes (or you check **Compile Help** on the *Export* dialog); **Mif2Go** runs the compiler after converting your document. Try compiling directly from Help Workshop to see if that is where the problem lies.

*OmniHelp
Loading...*

If you use **Mif2Go** to generate OmniHelp, then when you load *_myproj.htm* the browser displays only this message, and nothing else happens:

OmniHelp Loading...

This usually means the OmniHelp viewer files or control files are not in the same directory as the HTML files; see §10.2 [Setting up OmniHelp viewer control files](#) on page 342.

Everything where it belongs? Next: [Check for missing Microsoft components](#).

D.2.3 Check for missing Microsoft components

If FrameMaker crashes when you browse for a **Path** in the *Choose Project* dialog, check to make sure you have a copy of library file `msvc60.dll` in your Windows system directory. If this file is not present, or is dated before 29 August 2000, download a copy from:

<http://dl.dropbox.com/u/1868997/msvc60.dll>

Place this file in your Windows system directory (`\windows\system32` or, for 64-bit systems, `\windows\SysWOW64`) before you start FrameMaker again.

Microsoft components present? Next: [Check the Mif2Go User's Guide](#).

D.2.4 Check the Mif2Go User's Guide

Quickest way: use the **Search** feature of the HTML Help version, which is installed with **Mif2Go**. Search for words likely to relate to the problem; you might be able to solve it yourself.

No luck? Next: [Check path names, file names, and drive location](#).

D.2.5 Check path names, file names, and drive location

If the name of any path or file involved in the conversion contains *any* characters other than letters and numbers (such as spaces, dashes, or underscores), rename the path or move the file to eliminate them; see §1.1.2 [File, directory, and path names](#) on page 51. FrameMaker does not handle spaces in file names very well; **Mif2Go** does not care.

If all file and path names are free of non-alphanumeric characters and your conversion is still not producing any output, check whether your `%OMSYSHOME%` directory or any of your FrameMaker files are on a network drive. FrameMaker is noted for not running properly when files are on a network drive.

Another issue is “latency”, where a large and unpredictable delay occurs between a program's file request and the server's response. So you can get a “race” condition, where one instance of **Mif2Go** closes a file, and the next instance opens the same file. If the close is not completed, which involves numerous internal updates on the server, the open may be denied. The result will be broken links. You may get other results for other such race conditions.

If you are using a network drive, move your files to a local drive and try running the same conversion there before asking for support.

File names and paths valid, and the problem still exists? Next: [Check for file corruption](#).

D.2.6 Check for file corruption

A slightly corrupted FrameMaker file might cause hard-to-trace problems with **Mif2Go**. To rule out this possibility, try cleaning out any corruption by saving the offending file as MIF and replacing the binary version with the MIF version. **Mif2Go** can do this for you (even in the evaluation version), and report the results in the FrameMaker console window.

Clean a single file

To clean a FrameMaker binary file: with the file open and active, on the FrameMaker **File** menu choose **Wash Via MIF**. **Mif2Go** does the following:

1. Saves the file as MIF with extension `.tmb` (for Temporary MIF Backup), to avoid conflict with any existing MIF file of the same name.

2. Closes the file without saving.
3. Opens the .tmb MIF file.
4. Saves the .tmb MIF file as .fm, overwriting the original file.
5. Deletes the .tmb MIF file.

Note: FrameMaker version 11 seems to get as far as [Step 3](#) and then stop; and fails to process chapter files when run on a book.

Clean all files in a book

To clean all files in a FrameMaker book: with the book file active, hold down the Shift key, and on the FrameMaker **File** menu choose **Wash All Files in Book Via MIF**. **Mif2Go** processes each file in the book as described for **Wash Via MIF**, afterward closing any file that was not open in the first place.

Back-up copies are saved first

If you have set up FrameMaker to save back-up files, when you “wash” files via MIF, **Mif2Go** saves the previous .fm or .book file as *filename.backup.fm* or *filename.backup.book*.

File(s) decontaminated, and the problem still exists? Next: [Check for broken links \(HTML or XML output\)](#).

D.2.7 Check for broken links (HTML or XML output)

When you produce HTML or XML output from a FrameMaker book with `CheckLinks=Yes` (see §5.1.5 [Checking for broken links in HTML or XML output](#) on page 112), you might get an error message about broken links. If **Mif2Go** is unable to resolve any interfile links in your document, you will see a FrameMaker Book Error Log at the end of processing, and the FrameMaker console report will list the number of broken links.

Check ObjectIDs setting

Assuming you were able to successfully update the book before starting **Mif2Go**, one possible problem might be the following setting:

```
[HTMLOptions]
ObjectIDs = Referenced
```

If this setting is included in your project configuration file or in a referenced configuration template, change or override the setting as follows:

```
[HTMLOptions]
ObjectIDs = All
```

See §19.5.3 [Including ObjectID anchors as link targets](#) on page 620. If you are converting generated files, see also §13.8.2.2 [Including paragraph references](#) on page 445.

Convert again, retaining .ref files

In a few situations you might have to run a conversion twice (*without* deleting .ref files in between runs) before all links are resolved. The first pass creates or updates the necessary .ref files, and the second pass accesses the link information in those files. See §C.5 [Working with reference files for HTML or XML](#) on page 1027.

Check for relative vs. absolute paths

If links work locally but not when you move all the output files to another location, check whether your project includes any `[XrefFiles]` settings that specify absolute paths. These settings must specify relative paths; otherwise the links work only on the system where you ran the conversion. See §19.6.3 [Enabling links to renamed or relocated files](#) on page 622.

No broken links, and the problem still exists? Next: [Restart FrameMaker, reboot system](#).

D.2.8 Restart FrameMaker, reboot system

Do you close and then reopen FrameMaker between **Mif2Go** runs? This should be standard operating procedure; see §2.6 [Establishing a conversion environment](#) on page 74.

If you are using Windows 9x or Windows ME, also try rebooting the system.

The problem persists? Next: [Check your version of Mif2Go](#).

D.2.9 Check your version of Mif2Go

If you are using the evaluation version of **Mif2Go**, see §D.1 [Evaluation version is different](#) on page 1029.

If you are using a licensed version of **Mif2Go**, do the following:

1. In a text editor, open an output file (.htm or .rtf or .dita or .ent) that you created with **Mif2Go**, and find a line near the top of the file that shows the **Mif2Go** version and build numbers. The line you want looks like this:

```
HTML, DITA, <!-- generated by DCL filter dwhtml, Ver 4.0 m199 h280 -->
or DocBook:
RTF:          {\info {\doccomm DCL filter dwrtf, Ver 4.0 m199 r285}}
```

[Table D-1](#) on page 1034 lists the build numbers underlined in these examples.

2. If you are running **Mif2Go** from inside FrameMaker, check the build number of the **Mif2Go** plug-in, m2rbook.dll. Look for keyword PluginVersion in the [Setup] section of the project configuration file you were using when the problem occurred. **Mif2Go** updates the value of PluginVersion each time you run a conversion. For example:

```
[Setup]
PluginVersion=b112
```

[Table D-1](#) on page 1034 lists the build number shown in bold in this example.

To make sure you are using the most current build, check PluginVersion value bNNN against the version of m2rbookNNN.zip on the Omni Systems Web site:

<http://mif2go.com>

3. Go to the **Mif2Go** Web site:

<http://mif2go.com>

Navigate to **Downloads > Registered Software > Components**, and check the build numbers on the archived DLL files. For example:

<u>File</u>	<u>Size</u>	<u>Description</u>	<u>Last updated</u>
m2rbook 112 .zip	196k	Plugin main module	01-May-2012
drmif 208 .zip	177k	MIF input module	01-May-2012
dwrftf 295 .zip	251k	RTF output module	01-Jun-2010
dwhtml 289 .zip	421k	HTML/XML output module	01-Jun-2010

4. Compare the number on each DLL archive file with the build numbers you found in your output, as shown in [Table D-1](#).

Table D-1 Examples of build numbers for Mif2Go DLL files

Output type	DLL file	Build number:		
		Latest	Used	Current?
All	m2rbook.dll	112	b112	Yes
	drmif.dll	208	m208	Yes
HTML	dwhtml.dll	289	h284	No
RTF	dwrftf.dll	295	r295	Yes

5. If the build number on a DLL archive is higher than the corresponding build number in your output file (or in the configuration file), obtain and install the current update; see §1.4 [How to update Mif2Go](#) on page 61. Then try the conversion again.
6. If you think you have all the latest DLLs, but a build number in the output file does not agree, there might be an old copy somewhere on your system, typically in `\windows\system` or `\windows\system32`. Find and delete the old copy, then download an updated copy and unzip it in `%OMSYSHOME%\common\bin`.
7. If you still encounter the problem, check whether later beta versions of any DLLs are available on the **Mif2Go** Web site:
<http://mif2go.com>

Navigate to **Downloads > Registered Software > Beta Components**, and check the four-part numbers in the descriptions of the DLL files. The first two parts are the product version, third part the build number, and fourth is the beta version, zero for the released DLL, incremented for each beta build. For example:

<u>File</u>	<u>Size</u>	<u>Description</u>	<u>Last updated</u>
m2rbook.dll	572k	Plugin main module, 4.0.112.0	01-May-2012
drmfif.dll	412k	MIF input module, 4.0.208.0	01-May-2012
dwrftf.dll	588k	RTF output module, 4.0.297.2	02-Jun-2012
dwhhtm.dll	1052k	HTML/XML output module, 4.0.291.4	04-Jun-2012

8. Compare the third part of the number in each description with the build numbers you found in your output, as shown in [Table D-1](#). If the fourth part is greater than zero, and the problem is due to a defect in **Mif2Go**, the defect might have been corrected. See §1.4.3 [Try out Mif2Go beta executables](#) on page 62.

All DLLs up to date, and the problem still exists? Next: [Narrow down the problem](#).

D.2.10 Narrow down the problem

Make a copy of a FrameMaker file that produces the problem, then split the copy in two parts, and see if the problem happens with only one part. If so, keep splitting until you are down to a page or less; you might find something questionable in the file that you can fix to get it working.

Still no luck? See §D.3 [How to request help](#) on page 1035.

D.3 How to request help

Zip your files! **Do not send unzipped FrameMaker files to Omni Systems.**
Do not send files larger than 1 MB.

If you still encounter problems after following the steps in §D.2 [Things to check first](#) on page 1029, help us to help you, as follows:

- §D.3.1 [If the problem involves a crash](#) on page 1035
- §D.3.2 [Scope the problem](#) on page 1036
- §D.3.3 [Document the problem](#) on page 1036
- §D.3.4 [Package the problem](#) on page 1037
- §D.3.5 [Send the package to Omni Systems](#) on page 1037

D.3.1 If the problem involves a crash

If you are getting a Windows error message such as the following:

DCL NT console driver has encountered a problem and needs to close.

This means your **Mif2Go** conversion caused a crash. If the crash happens when **Mif2Go** is converting a FrameMaker index, see §5.5 [Converting FrameMaker-generated files](#) on page 124; you might need to add character formats to page numbers. Otherwise, try the following debugging options:

```
[Options]
; NoNameDel = No (default),
; or Yes (prevent deallocation of name memory)
NoNameDel=Yes
; NoMemDel = No (default) or Yes (prevent deallocation of all memory)
NoMemDel=Yes
```

First set NoNameDel=Yes. If the conversion still causes a crash, try setting NoMemDel=Yes. Your conversion might run to completion with one or the other of these options; in any event, document the result, so Omni Systems programmers can investigate.

Note: When you set either of these options to Yes, memory deallocation is prevented only while `dcl.exe` is running; at the end of that (usually brief) process, all memory used by **Mif2Go** is always freed; no memory leaks occur.

Next: [Scope the problem](#).

D.3.2 Scope the problem

Use the Configuration Manager (see §4.2 [Editing files with the Configuration Manager](#) on page 91) to check the configurations in use. If any settings in “local” configuration files in `%OMSYSHOME%\m2g\local` subdirectories might affect the test case, copy those settings into your project configuration file, and see if that fixes the problem. Otherwise, consider the following questions:

- Do you get the same result each time you try, or does the result vary?
- If you have machines with other operating-system versions available, does the same thing happen on all of them?
- Does it happen with all source files, or only some? If only some, do the problem files have something in common that other files do not?

Next: [Document the problem](#).

D.3.3 Document the problem

Write an e-mail message that contains the following information:

- A brief description of the problem, including answers to questions in §D.3.2 [Scope the problem](#) on page 1036.
- Operating-system name and version; for example, Windows 7 X64.
- Amount of memory on your machine; for example, 2 GB.
- FrameMaker version (click **Help > About FrameMaker...**); for example, 5.5.6p145.
- Browser name and version, if the problem occurs when you generate HTML; for example, Firefox 3.5.

If **Mif2Go** crashed, we also need to know just what information was displayed, at the time of the crash, on the status bar at the bottom of the book window or file window.

Next: [Package the problem](#).

D.3.4 Package the problem

If you have placed settings in a file in %OMSYSHOME%\m2g\local\config*, copy those settings to your project configuration file before you do the following.

Create a .zip file *smaller than 1 MB* that contains the following files:

FrameMaker file(s)	The smallest fragments that yield the problem; see Step D.2.10 in §D.2 Things to check first on page 1029. <i>No unzipped files.</i>
MIF file(s)	The .mif file(s) Mif2Go creates from the above .fm file(s).
FrameMaker conversion template	If your project uses a conversion template, and the problem is a display problem, include the conversion template.
Book file	If Mif2Go crashed while you were converting a book.
Output file(s)	Whatever output (if any) shows the undesired result.
Log file	Located by default in your project directory; see §5.2 Logging conversion events on page 115.
Configuration file(s)	Your project configuration file, plus any chapter-specific configuration file used by the problem FrameMaker file.
Configuration templates	Include all configuration files and templates in every chain that might affect the result, except the distribution templates. However, if you have placed settings in a file in %OMSYSHOME%\m2g\local\config*, copy those settings to your project configuration file before you create the package.
Macro libraries	If the problem file uses Mif2Go macros located in library files.
CSS file(s)	If your project uses CSS (HTML output only), and the problem is a display problem.
Mif2Go project file	<i>MyDoc.prj</i> , located in the input directory; see §C.1 Locating document and conversion files on page 1019.
Mif2Go FileID file	<i>mif2go.ini</i> , located in the input directory; see §C.1 Locating document and conversion files on page 1019.
Help project file	If the problem occurs when you generate one of the following: <ul style="list-style-type: none"> • WinHelp: <i>MyDoc.hpj</i>. • HTML Help: <i>MyDoc.hhp</i>. • JavaHelp or Oracle Help for Java: <i>MyDoc.hs</i>.
Graphics files	Any external graphics referenced by the problem-file fragment.

Zip your files! *Do not send unzipped FrameMaker files; they are nearly always corrupted in transit.*

We need to be able to unzip and run your test case at Omni Systems with no further ado.

Finally: [Send the package to Omni Systems](#).

D.3.5 Send the package to Omni Systems

Attach the .zip file you created in §D.3.4 [Package the problem](#) on page 1037 to the e-mail message you wrote in §D.3.3 [Document the problem](#) on page 1036, and send it to:

support@omsys.com

Generally you will receive a response within one business day; sometimes within an hour. If you have not heard from Omni Systems after one business day, send another e-mail message (*without attachments*) to inquire.

Zip your files! *Do not send unzipped FrameMaker files to Omni Systems.
Do not send files larger than 1 MB.*

E DITA <bookmeta> template

This section presents a template for the DITA version 1.1 <bookmeta> element. This template is available in your **Mif2Go** distribution directory as file `bookmeta.xml`. Copy the file to your project directory, delete elements you do not need, substitute values for elements you do need, and provide a reference in your project configuration file to the resulting customized template file; see §16.3.3 [Specifying <bookmeta> information](#) on page 549.

```
<bookmeta>

<linktext>text description for xrefs</linktext>
<searchtitle>text description for search</searchtitle>
<shortdesc>detailed text description</shortdesc>

<author type="creator">name for the first author</author>
<author type="contributor">name for additional author</author>
OR
<authorinformation>
  <personinfo>
    <namedetails>
      <personname>
        <firstname>text</firstname>
        <middlename>text</middlename>
        <lastname>text</lastname>
        <generationidentifier>text</generationidentifier>
        <otherinfo>text</otherinfo>
      </personname>
    </namedetails>
    <contactnumbers>
      <contactnumber>text</contactnumber>
    </contactnumbers>
    <emailaddresses>
      <emailaddress>text</emailaddress>
    </emailaddresses>
  </personinfo>
  <organizationinfo>
    <namedetails>
      <organizationnamedetails>
        <organizationname>text</organizationname>
        <otherinfo>text</otherinfo>
      </organizationnamedetails>
    </namedetails>
    <addressdetails>
      <thoroughfare>text</thoroughfare>
      <locality>
        <localityname>text</localityname>
        <postalcode>text</postalcode>
      </locality>
      <administrativearea>text</administrativearea>
      <country>text</country>
    </addressdetails>
  </organizationinfo>
</authorinformation>

<source href="URL/of/source">text</source>

<publisherinformation>
```

```

<person>text</person>
<organization>text</organization>
<printlocation>text</printlocation>
<published>
  <person>text</person>
  <organization>text</organization>
  <publishtype name="" value=""/>
  <revisionid>text</revisionid>
  <started><year>text</year><month>text</month><day>text</day></started>

<completed><year>text</year><month>text</month><day>text</day></completed>
>
  <summary>text</summary>
</published>
</publisherinformation>

<critdates>
  <created date="2001-06-12"/>
  <revised modified="2001-08-20"/>
</critdates>

<permissions view="attributes used for dita filtering"/>
<audience type="" job="" experiencelevel="" name=""/>
<category>text</category>

<keywords>
<keyword>text</keyword>
</keywords>

<prodinfo>
  <prodname>text</prodname>
  <vrmlist>
    <vrmlist version="" release="" modification=""/>
  </vrmlist>
  <brand>text</brand>
  <series>text</series>
  <platform>text</platform>
  <prognum>text</prognum>
  <featnum>text</featnum>
  <component>text</component>
</prodinfo>

<othermeta name="" content=""/>
<resourceid id="" appname=""/>

<bookid>
  <bookpartno>text</bookpartno>
  <edition>text</edition>
  <isbn>text</isbn>
  <booknumber>text</booknumber>
  <volume>text</volume>
  <maintainer>
    <person>text</person>
    <organization>text</organization>
  </maintainer>
</bookid>

<bookchangehistory>
  <reviewed>
    <person>text</person>
    <organization>text</organization>
    <publishtype name="" value=""/>

```

```

    <revisionid>text revision number</revisionid>
    <started><year>text</year><month>text</month><day>text</day></started>

<completed><year>text</year><month>text</month><day>text</day></completed>
>
    <summary>text</summary>
</reviewed>
<edited>
    (same)
</edited>
<tested>
    (same)
</tested>
<approved>
    (same)
</approved>
<bookevent>
    <bookeventtype name="event"/>
    (same)
</bookevent>
</bookchangehistory>

<bookrights>
    <copyrfirst><year>text</year></copyrfirst>
    <copyrlast><year>text</year></copyrlast>
    <bookowner>
        <person>text</person>
        <organization>text</organization>
    </bookowner>
    <bookrestriction value="" />
    <summary>text</summary>
</bookrights>

<data name="" value="" href="">text</data>

</bookmeta>

```


F Content model configuration

This section provides an annotated list of configuration sections, keywords, and acceptable values for settings in content-model configuration files.

See also:

§32 [Working with content models](#) on page 905

```
; ContentModel.txt describes sections used in DITASpecial.ini files,  
; such as DITAconcept11.ini, as they are supported in dwhtml.dll h283.  
; Most of it also applies to DocBook content model files; differences  
; are marked in the descriptions below.
```

```
[Topic]  
;ModelName = name of type (usually a built-in) to be replaced after  
; this file loads, effective only when this file is specified in  
; [DITAContentModels] or [DocBookOptions]ContentModel in mif2html.ini;  
; overrides the default use of the filename (without "DITA").  
ModelName=concept
```

```
;  
; TopicRoot = name of root element in the DITA or DocBook file for  
; this type.  
TopicRoot=concept
```

```
; These two are DITA-only, not for DocBook:  
; TopicStart = name of element that starts topic, such as "glossterm"  
; (for glossary) or "title" (for every other type). When the Frame  
; format mapped to this element in [DITATags] is also mapped to  
; level 1 in [DITALEvels], that format always starts a new topic.  
TopicStart=title  
; TopicBody = name of its body element, such as conbody for concept.  
TopicBody=conbody
```

```
; PrologDType = PUBLIC name used in DOCTYPE header, double quotes  
; are required.  
PrologDType="-//OASIS//DTD DITA Concept//EN".  
; PrologDTD = SYSTEM name, such as "concept.dtd", can include a path,  
; double quotes are required.  
PrologDTD="http://docs.oasis-open.org/dita/v1.1/CD01/dtd/concept.dtd".  
;  
;TopicDerivation = name of type from which it is derived, either one of  
; the defined types (topic, concept, task, reference, glossary, or map)  
; or another specialized type for which an .ini is available. Needed  
; iff the description in the rest of the sections is additive rather  
; than complete in itself; omitted otherwise. Not used for .inis that  
; were generated by dtd2ini, which are always complete.  
TopicDerivation=topic  
;  
;DumpToFile = name with optional path of file in which to dump the  
; tagset information (including error lists) after loading, for debug;  
; default none, meaning don't dump. If the tagset is used more than  
; once in processing the Frame file, it is dumped only the first time.  
DumpToFile=concept2dump.txt
```

```
; For DITA working examples of the following sections, see the files  
; DITAtopic*.ini, DITAconcept*.ini, DITAtask*.ini, DITAreference*.ini,  
; DITAglossary*.ini, DITAbookmap*.ini and DITAmapping*.ini, where * is  
; 10 for version 1.0 and 11 for version 1.1. DocBook examples are
```

```
; docbook45b.ini (book as root) and docbook45a.ini (article as root).
```

[TopicParents]

```
; Element name = possible parents. All elements other than the topic
; type itself, and its body type, must be listed on the left here.
; The two reserved parent names are "Any" (any parent is acceptable,
; mainly for inline elements) and "No" (for any elements present in
; the derived-from type that are excluded from this type). If there
; is more than one possible parent, they must be defined as a single
; set, and listed in [ElementSets] below.
```

[ElementSets]

```
; Name for set = list of elements. This allows grouping of elements
; for use on the right side of [TopicParents] and [TopicFirst], so
; that the same set of parents can be used for more than one element.
; The lists of elements on the right here can include sets too, as
; building blocks. The sets are roughly equivalent to the parameter
; entities used in the DITA DTDs. Set names must start with "*", and
; sets can include other sets. Included sets should preferably be
; defined above the sets including them; in any case, circular set
; references (set A includes set B and set B includes set A, directly
; or indirectly) will not work.
```

[ElementTypes]

```
; Element name = list of properties: Block or Inline, Text, and
; Preform; default is Block without Text. The Block and Inline
; properties determine whether returns are inserted before start
; tags and after end tags. The Text property determines whether
; an attempt is made to wrap any invalid text (in an element that
; does not allow Text) in a valid container element, like <ph>.
; Preform determines whether whitespace within the element is
; retained as is; those elements are always block and allow text.
; For example:
para=Block Text
ph=Inline Text
section=Block
menucascade=Inline
codeblock=Block Text Preform
```

[TopicLevels]

```
; Element name = required level in topic, used only for elements that
; must be at a specific level, such as shortdesc, prolog, body, and
; related-links at level 1, and example and metadata at level 2.
; The content models generated by dtd2ini name only level 1 elements.
```

[TopicFirst]

```
; Child element = parents, where child must be the first child of the
; specified parents; if child is not first, the current parent is
; closed and a new instance of it is started. Used mainly for lists,
; as in dt=dlentry and pt=plentry, and for title=Any. To add more
; than one parent when Any won't do, specify them in [ElementSets].
```

```
; The remaining sections are used for DITA only, not DocBook:
```

[TopicTables]

```
; Table name = name of section that describes it below. All supported
; by this topic type (other than those defined in the type derived from)
; are defined here. Note that multiple named tables can define variants
; of the same DITA TableType; the name is purely a Mif2Go identifier.
; A name can be undefined in a derived topic type by setting name=No.
; Since dtd2ini does not generate these sections, they must either be
```



```

; included in dtd2ini.ini as [AddedSections], or added to the generated
; content model .ini manually after dtd2ini produces it.

; These examples of table descriptions show all available table settings.

[PropertyTable]
TableType=properties
; ColCountMax default is 0, for unlimited, as for simpletable
ColCountMax=3
;
; HeadRowMax default is 0, for unlimited head rows.
HeadRowMax=1
; HeadRow is applied only to the initial rows, iff they are head
; rows in the Frame file.
HeadRow=prophead
; All cells are used; to omit some, define another table name with
; fewer columns but the same TableType.
HeadCell1=proptypehd
HeadCell2=propvaluehd
HeadCell3=propdeschd
;
Row=property
Cell1=proptype
Cell2=propvalue
Cell3=propdesc

[SimpleTable]
TableType=simpletable
HeadRowMax=1
HeadRow=sthead
Row=strow
Cell=stentry

[ComplexTable]
TableType=complex
; TableTitle default is No, for no title.
TableTitle=Yes
; TableDesc default is no desc.
TableDesc=desc
; TableGroup default is no group'
TableGroup=tgroup
; ColSpec default is no column specs
ColSpec=colspec
; The next three items are all colspec attributes
ColNum=colnum
; ColSpecName is required if ColSpanNames=Yes or ColName is
; used, below. It is created using ColNamePrefix, below.
ColSpecName=colname
ColWidth=colwidth
;
; HeadGroup default is no group, use head rows only.
HeadGroup=thead
; HeadRow Default is same row element as for body.
HeadRow=hrow
; HeadCell default is same cell element as for body
HeadCell=hentry
;
; BodyGroup default is no group, use body rows only.
BodyGroup=tbody
Row=row
Cell=entry
;

```

```
; RowSpan is a cell attribute name; default is no rowspan.  
RowSpan=morerows  
;  
; ColSpanNames default is true to use names, false uses count.  
ColSpanNames=Yes  
; The next four settings are all cell attributes.  
; ColSpanCount is count of cells spanned, if ColSpanNames=No.  
ColSpanCount=span  
; ColSpanStart is ref to first colspec name if ColSpanNames=Yes.  
ColSpanStart=namest  
; ColSpanEnd is ref to last colspec name if ColSpanNames=Yes.  
ColSpanEnd=nameend  
; ColName is ref to single colspec name for non-spanning cells.  
ColName=colname  
; ColNamePrefix is for colspec names, default col as in DITA-OT.  
ColNamePrefix=col  
; CellAlign default is No, when Yes use align and valign attrs.  
CellAlign=Yes
```

[End]

RTF keyword index

ABCDEFGHIJKLMNOPQRSTUVWXYZ

A

AddCntFileName, [HelpContents] keyword 291
AKey, [HelpStyles] format property 222, 269, 285
ALink, [MarkerTypes] property 221, 839
AllowLiningOverrides, [HelpOptions]
 keyword 254
Altura, [HelpOptions] keyword 247
[AnumCodeAfter], code after paragraph autonumber
 placement properties 823
 subject to configuration overrides 926
[AnumCodeBefore], code before paragraph auto-
 number
 placement properties 823
 subject to configuration overrides 926
AppliedTemplateFlags, [Setup] keyword 864
 change template options 866
 set-up option 81
ApplyTemplateFile, [Setup] keyword 864
 change template options 866
 set-up option 81
Archive*, [Automation] keywords:
 ArchiveCommand 973
 activated by WrapAndShip 956
 ArchiveEndParams 974
 ArchiveExt 974
 ArchiveName 974
 ArchiveStartParams 973
 ArchiveVer 974
AskForUserVars, [Automation] keyword 942
AutoBrowse, [HelpBrowse] keyword 292
[Automation]
 default values in local_omsys.ini 59
 export options and settings 84
 options determined at run time 863
 produce deliverables 956
 system commands 938, 939
 user variables 942

B

BackMode, [Graphics] keyword 903
[BctFileHeads], WinHelp section 289

[BitmapChars], WinHelp section 255
BitmapDPI, [Graphics] keyword
 override with a *Config marker 895
 rescale bitmap graphics 898
BitmapFlip, [Graphics] keyword 899
BitmapFont, [BitmapChars] keyword 255
BMPsForDingbats, [HelpOptions] keyword 256
BookmarkIXRanges, [WordOptions] keyword 148
Bottom, [Inserts] Word keyword 822
Browse, [HelpStyles] format property 269
[BrowsePrefix], WinHelp section 293
[BrowseStart], WinHelp section 293
Build, [HelpStyles] format property 269
BulletFile, [Graphics] keyword 256
Bullets, [HelpOptions] keyword 256

C

CaselessMatch, [Options] keyword 113
 case sensitivity of FileIDs 121
[CharStyle*] sections
 [CharStyleCode*] sections
 all subject to configuration overrides 926
 [CharStyleCodeAfter] 823
 [CharStyleCodeBefore] 823
 [CharStyleCodeEnd] 823
 [CharStyleCodeReplace] 823
 [CharStyleCodeStart] 823
CharStylesUsedInText
 [HelpOptions] keyword 254
 [WordOptions] keyword 163
CharTags, [WordOptions] keyword 163
ClipLimit, [Graphics] keyword 902
ClipType, [Graphics] keyword 902
Cnt*, [HelpContents] keywords:
 CntBase
 set-up option 245
 288
 CntBStyleText 290, 291
 CntMainWindow 291
 CntName 289
 set-up option 245

- CntStartFile 289
- CntTitle 289
 - set-up option 245
- CntTopHead 289
- CntTopic 289
 - set-up option 245
- CntType 208, 288
- Code, [MarkerTypes] property 839
- Code*, [HelpStyles] and [WordStyles] format properties
 - CodeAfter 804, 823
 - CodeAfterAnum 823
 - CodeBefore 804, 823
 - CodeBeforeAnum 823
 - CodeEnd 823
 - CodeReplace 823
 - CodeStart 804, 823
 - CodeStore 796, 804
- CodePage, [Defaults] keyword 148
- CompileHelp, [Automation] keyword 250
 - compile WinHelp project 971
 - determined at run time 863
 - export option 84
 - set up WinHelp project 245, 248
- Compiler, [HelpOptions] keyword 250
- CompressRasters, [Graphics] keyword 899
- [ConditionsShown], apply FrameMaker conditions 123
- Config, override configuration settings
 - [HelpStyles] format property 931
 - [MarkerTypes] property 839
 - [WordStyles] format property 931
- Configs, [Templates] keyword 851, 859, 862
 - chain of templates 863
 - precedence of settings 919
- Contents, [HelpStyles] format property 209, 269
- ConvertVariables, [Setup] keyword 157
 - convert system variables to text 114
 - set-up option 81
- CopyAfterFiles, [Automation] keyword 964
- CopyAfterFrom, [Automation] keyword 964
- CopyBeforeFiles, [Automation] keyword 960
- CopyBeforeFrom, [Automation] keyword 960
- CopyGraphicsFrom, [Automation] keyword 966
 - activated by CompileHelp 972
 - activated by WrapAndShip 956
- CopyOriginalGraphics, [Automation] keyword 965

D

- [Defaults] 147, 166, 170
 - subject to configuration overrides 925
- [DefaultUnicodeFonts], for FrameMaker 8 Unicode 169
- DefBrushType, [Graphics] keyword 899
- DefFont, [Graphics] keyword 902
- DefFSize, [Graphics] keyword 902
- DefTabWidth
 - [HelpOptions] keyword 253
 - [WordOptions] keyword 165
- Delete
 - [HelpStyles] format property 269
 - [MarkerTypes] property 839
 - [XrefStyles] format property 260
- Delete, [WordStyles] format property 174
- DeleteExistingDCL, [Setup] keyword 112
- DeleteExistingMIF, [Automation] keyword 111
 - activated by CompileHelp 972
- Digits, [HelpBrowse] keyword 292
- DisambiguateIndex, [HelpOptions] keyword 287
 - dependencies 287
- Document, [Templates] keyword 854, 859, 860

E

- EditorFileName, [Logging] keyword 115
- EmbedBMPsInWMFs, [Graphics] keyword 264, 873, 886, 890
- EmbedEqsInWMFs, [HelpOptions] keyword 138
- EmbedEqsInWMFs, [WordOptions] keyword 138
- EmptyFrames
 - [HelpOptions] keyword 895
 - [WordOptions] keyword 895
- EmptyGraphPath, [Automation] keyword
 - activated by CompileHelp 972
- EmptyOutputDir, [Automation] keyword 958
 - activated by CompileHelp 972
 - dependencies 959
 - when effective 958
- EmptyOutputFiles, [Automation] keyword 958
 - activated by CompileHelp 972
 - when to include 959
- EmptyWrapPath, [Automation] keyword 962
 - activated by CompileHelp 972

- dependencies 968
- [End], dummy section to end settings 107
- [End], dummy section to replace
 - [MacroVariables] 788
- EndFtnWithSpace, [HelpOptions] keyword 248
- EpsiUsage, [Graphics] keyword 876
- EqHorAdjust, [HelpOptions] keyword 138
- EqHorAdjust, [WordOptions] keyword 138
- EqSuffix, [HelpOptions] keyword 137
- EqSuffix, [Options] keyword 726
- EqSuffix, [WordOptions] keyword 137
- EquationExportDPI, [Setup] keyword 137
 - graphic format and resolution 884
- EquationFrameExpand, [Setup] keyword 137
- EqVertAdjust, [HelpOptions] keyword 138
- EqVertAdjust, [WordOptions] keyword 138
- ExactLineSpace, [WordOptions] keyword 170
- Export*, [GraphExport] keywords:
 - ExportBmpFiles 881
 - ExportCdrFiles 881
 - ExportEpsFiles 881
 - ExportGifFiles 881
 - ExportJpgFiles 881
 - ExportNameChars 134
 - ExportNumDigits 134
 - ExportPctFiles 881
 - ExportPcxFiles 881
 - ExportPngFiles 881
 - ExportRfFiles 881
 - ExportTifFiles 881
 - ExportWmfFiles 881
 - ExportWpgFiles 881
 - set-up options 1011
- ExtendHelpNoScroll, [HelpOptions]
 - keyword 270, 271
- ExternalXrefs, [WordOptions] keyword 148, 179

F

- FieldHyper, [WordOptions] keyword
 - (*deprecated*) 176
- File, [HelpStyles] format property 269
 - [FileIDs]
 - deprecated for main configuration file 122
 - mif2go.ini section 121
- FileNames, [Graphics] keyword 891
- reference WinHelp hypergraphics 275
- replace file extensions 131
- substitute files 188, 189, 891, 892
- synchronize settings 969
- FilePaths, [Graphics] keyword 891
 - for already converted files 893
 - for referenced graphics 188
 - omit for exported graphics 189
 - omit for unconverted graphics 189
 - reference WinHelp hypergraphics 275
 - replace EPSI graphics 877
 - substitute files 891, 892
 - synchronize settings 969
- FileSuffix, [Setup] keyword 147, 1008
 - export option 84
 - set-up option 1007
- FirstFooter, [Inserts] Word keyword 822
- FirstHeader, [Inserts] Word keyword 822
- FixMacroQuotes, [Macros] keyword 790
 - [FontEncoding], for print RTF 169, 255
- FontName, [Defaults] keyword 166
 - [Fonts], remap fonts 166
- FontSize, [Defaults] keyword 166
 - [FontTypes]
 - for print RTF 168
 - for WinHelp 255
- FontWidth, RTF [Defaults] keyword 166
 - [FontWidths] 166
- Footer, [Inserts] Word keyword 822
- Footnotes
 - [HelpOptions] keyword 258
 - [WordOptions] keyword 154
- FootnoteSeparator
 - [HelpOptions] keyword 258
- FootnoteSpace, [HelpOptions] keyword 248
- ForceBmc, [HelpOptions] keyword 247
- ForceSideHeadPos, [WordOptions] keyword 159
- ForceTableLineBreaks, [Tables] keyword 262
- FrameBorders, [Graphics] keyword 900
- FrameDefaultFontName, [Graphics]
 - keyword 901
- FrameDefaultFontSize, [Graphics]
 - keyword 901
- FrameEndPara, [WordOptions] keyword 174
- FrameExactHeight, [Graphics] keyword 903
- FrameStyle
 - [HelpOptions] keyword 264

[WordOptions] keyword 191

FrBorders, [Graphics] keyword 900

G

GraphCopyFiles, [Automation] keyword 966
activated by CompileHelp or FTSCCommand 972

[GraphExport]

export embedded graphics 880
export images from OLE objects 882
name exported graphics files 134
set-up options and settings 1011
subject to configuration overrides 925

[GraphFiles], replace graphics files 131, 188, 189, 891

reference WinHelp hypergraphics 275
synchronize graphics settings 969

GraphicExportDPI, [Setup] keyword

FrameMaker export filters 130, 721, 884

GraphicExportFormat, [Setup] keyword

BMP instead of WMF graphics 873
FrameMaker export filters 130
graphic output format 884

GraphicNameDigits, [Setup] keyword 134, 885

[Graphics]

background 903
bitmaps
compress 899
embed 886
reorient 899
rescale 899

borders 899, 900

bullets 256

EPS 876

file extension 890

file names 135, 247

font 902

omit 895

options determined at run time, *listed* 863

scale 898

subject to configuration overrides 925

text 901–903

GraphicsFirst, [Setup] keyword 132, 885

export master- and reference-page graphics 885
process only graphics 88

[GraphLineStyle], print RTF section 900

GraphText, [Graphics] keyword 902

Green, [HelpStyles] format property 269, 281

GrVertAdjust, [Graphics] keyword 903

H

Header, [Inserts] Word keyword 822

HeadFoot, [WordOptions] keyword

convert to WordPerfect 154

position header and footer text 156

[Help*Styles], WinHelp sections

all subject to configuration overrides 282

[HelpBrowsePrefixStyles] 293

[HelpCntStyles]

basic conversion options 248

understand level numbers 209, 290

[HelpJumpFileStyles] 269, 283

[HelpKeywordStyles] 286

[HelpMacroStyles] 270, 284

[HelpRefStyles] 282

[HelpSuffixStyles] 270, 282

[HelpTitleSufStyles] 270, 271

[HelpTopicBuildStyles] 269

[HelpWindowStyles] 225, 270, 278

[HelpBrowse] 292, 293

subject to configuration overrides 925

[HelpContents] 288–292

set-up options and settings 245

subject to configuration overrides 925

HelpCopyDate, [HelpOptions] keyword 250

HelpCopyright, [HelpOptions] keyword 250

HelpLineBreak, [HelpOptions] keyword 272

[HelpOptions]

cross references 259, 260, 261

equations 137

footnotes 248, 258

formats

character 254

paragraph 252–253

removing 257

graphic text 902

graphics 874, 877, 895, 903

hotspots 281

index 208, 213, 287, 288

links 277

markers 277

ObjectIDs 266, 325

options determined at run time, *listed* 863

page and section breaks 247, 249

platforms 247

remove Word markers 114

special characters 256

subject to configuration overrides 925

tables 261, 262, 263

titles 271, 272

[HelpReplacements] 258, 266
 subject to configuration overrides 927

HelpSectionBreaks, [HelpOptions]
 keyword 249
 platform differences 247

[HelpStyles]
 “A” footnotes 222
 ALinks and keywords 286
 basic properties 248
 hotspots 275
 replace content 257, 266, 325
 subject to configuration overrides 927

HelpTabLimit, [HelpOptions] keyword 253

[HelpXrefFiles], cross references 260

HFFramed, [WordOptions] keyword 155
 convert to WordPerfect 154

HFGap, [WordOptions] keyword 155

HFVertAdjust, [WordOptions] keyword 156

Hide, [WordStyles] format property 173

HideWhiteText, [WordOptions] keyword 173

HistoryFileName, [Logging] keyword 115

HPJFileName, [HelpOptions] keyword 248

HyperHelp, [HelpOptions] keyword 247

I

IDAttrName, [WordOptions] keyword 135

IDFileName, [Setup] keyword 120, 1027
 determined at run time 863

IDRefAttrName, [WordOptions] keyword 135

IdxColon, [HelpOptions] keyword 213, 287

ImportDocProps, [Setup] keyword 865

ImportGraphics, [GraphExport] keyword
 export embedded graphics 133, 880
 export OLE objects 881
 set-up option 1011

Index
 [HelpOptions] keyword 208
 [WordOptions] keyword 195

IndexRanges, [HelpOptions] keyword 213

[Inserts], insert code at predefined locations 822
 subject to configuration overrides 925

IXnewlinkPrefix
 [HelpOptions] keyword 267

J

JumpHot, [HelpStyles] format property 269

JumpTarget, [HelpStyles] format property 269

K

KeepAnchorParagraphs, [WordOptions]
 keyword 171

KeepCompileWindow, [Automation] keyword 251
 activated by CompileHelp 972

KeepID, [WordStyles] format property 183

KeepIXMarkerIDs
 [HelpOptions] keyword 288
 [WordOptions] keyword 178

KeepSectBreaks
 [HelpOptions] keyword 249
 [WordOptions] keyword 153

Key, [HelpStyles] format property 269, 286

KeywordLimit, [HelpOptions] keyword 212

L

Language, [Defaults] keyword 147

LeftFooter, [Inserts] Word keyword 822

LeftHeader, [Inserts] Word keyword 822

LineSpacing, RTF [Defaults] keyword 170

Local, [HelpStyles] format property 269

LocalConfigPath, [Setup] keyword
 set-up option 60

LockHyper, [WordOptions] keyword 178, 194

LockXrefs, [WordOptions] keyword 175, 194

LogAuto, [Automation] keyword 956

LogDebug, [Logging] keyword 116

LogErrors, [Logging] keyword 115

LogFileName, [Logging] keyword 115
 [Logging] conversion events 115

LogInfo, [Logging] keyword 116

LogIniChains, [Logging] keyword 116

LogQuerys, [Logging] keyword 116

LogWarnings, [Logging] keyword 116

M

Macro, [HelpStyles] format property 269, 270

MacroHot, [HelpStyles] format property 284

MacroNestMax, [Macros] keyword 792, 816

Macros, [Templates] keyword 793, 851

[Macros]

- debug 820
- loop-control limits 816
- remove implicit line breaks 789

[MacroVariables] 797

- create a macro variable 796

MacroVarNesting, [Macros] keyword 798

MakeBookMIF, [Setup] keyword 1007

- file extensions for MIF output 1008
- include book file in MIF output 1007
- set-up option 1007

MakeCombinedCnt, [HelpOptions] keyword 248

- determined at run time 863
- export option 84
- set-up option 245

MakeRef, [HelpStyles] format property 270, 282

- for pop-up graphics 265

[Markers], invent and clone marker types 139, 836, 837

MarkerType11, [HelpOptions] keyword 277

[MarkerTypeCodeAfter] 843

[MarkerTypeCodeBefore] 843

[MarkerTypeCodeReplace] 843

[MarkerTypes], marker-type properties 838

MergeStradCells

- [Table] keyword 186
- [Tables] keyword 262

Metafiles

- [HelpOptions] keyword 874
- [WordOptions] keyword 874

MetaNameChars, [Graphics] keyword 135

MetaNumDigits, [Graphics] keyword 135

MIFBookSuffix, [Setup] keyword 1008

- set-up option 1007

MoveArchive, [Automation] keyword 976

N

NameGraphics, [Graphics] keyword 192

NameUndefinedMacros, [Macros] keyword 820

NameUndefinedMacroVars, [Macros]

- keyword 820

NameWMFsAsBMPs, [Graphics] keyword 890

accommodate platform differences 247

NoBlankFirstGTLine, [Graphics] keyword 900, 903

NoMemDel, [Options] keyword 1036

NoNameDel, [Options] keyword 1036

NoScroll, [HelpStyles] format property 270

NoSeeAlso

- [HelpOptions] keyword 288
- [WordOptions] keyword 196

NoSymMap, [WordOptions] keyword 172

NoTitle, [HelpStyles] format property 270

- for pop-up topics 268

NoXrefPopups, [HelpOptions] keyword 276

NoXScroll, [HelpStyles] format property 270, 271

- for pop-up topics 268

O

ObjectIDs

- [HelpOptions] keyword 249, 266
- [WordOptions] keyword 175

OccludedTabs, [WordOptions] keyword 165

OmitMacroReturns, [Macros] keyword 789

OnlyAuto, [Automation] keyword 977

[Options]

- debug 1036
- equations 138, 726
- for cases, spaces, and wildcards 113
- for conversion debugging 1036
- for tabs 165
- subject to configuration overrides 925

OrigExtForMIF, [Setup] keyword 1008

- set-up option 1007

P

PageBreaks

- [HelpOptions] keyword 249
- [WordOptions] keyword 152

PageColGap, [WordOptions] keyword 153

PageColumns, [WordOptions] keyword 153

ParaLink

- [HelpStyles] format property 138, 270, 275
- [WordStyles] format property 182

ParaLink, [HelpStyles] format property 269

ParaSpace, [WordOptions] keyword 171

[ParaStyle*] sections
 [ParaStyleCode*] sections
 all subject to configuration overrides 927
 [ParaStyleCodeAfter] 823
 [ParaStyleCodeBefore] 823
 [ParaStyleCodeEnd] 823
 [ParaStyleCodeReplace] 823
 [ParaStyleCodeStart] 823
 PicScale[WordOptions] keyword 192
 PluginVersion, [Setup] keyword 1034
 determined at run time 863
 Pop*, [HelpStyles] format properties
 PopContent 269, 281, 282
 PopHot 269, 281
 PopOver 269, 270, 274, 275, 282
 Prefix, [HelpBrowse] keyword 292
 PrevRef, [HelpStyles] format property 270, 282
 for pop-up graphics 265
 PrjFileName, [Setup] keyword 1026
 determined at run time 863

Q

Quotes
 [HelpOptions] keyword 256
 [WordOptions] keyword 172

R

RasterBorders, [Graphics] keyword 899
 Refer, [HelpStyles] format property 270, 282
 RefFrameDefFormat
 [HelpOptions] keyword 253
 [WordOptions] keyword 162
 [RefFrameFormats], reference frames 162
 RefFrames
 [HelpOptions] keyword 253
 [WordOptions] keyword 162
 RemoveGraphics, [Graphics] keyword 895
 RemoveUnusedFonts
 [HelpOptions] keyword 257
 [WordOptions] keyword 170
 RemoveUnusedStyles
 [HelpOptions] keyword 257
 [WordOptions] keyword 163
 RemoveWordTocMarkers, [HelpOptions]
 keyword 114
 RepeatMax, [Macros] keyword 817

Replace
 [HelpStyles] format property 257, 266, 270
 [WordStyles] format property 174
 Replace, [HelpStyles] format property 269
 ReplaceFrameVars, [Macros] keyword 123
 Resume, [HelpStyles] format property 265, 268,
 269, 270, 275
 RevProt, [WordOptions] keyword 194
 RevTrack, [WordOptions] keyword 194
 RightFooter, [Inserts] Word keyword 822
 RightHeader, [Inserts] Word keyword 822
 RMarginTabs, [WordOptions] keyword 165
 RTFConfig
 [HelpStyles] format property 931
 [MarkerTypes] property 839
 [WordStyles] format property 931
 RunfmDiagnostics, [Automation] keyword 957,
 988
 RunInHeads
 [HelpOptions] keyword 252
 [WordOptions] keyword 160
 RunInHeads, [HelpOptions] keyword
 for Help systems 203

S

Scope, [Templates] keyword 855, 861
 Scroll, [HelpStyles] format property 270, 271
 for pop-up topics 268
 SeqAnums, [WordOptions] keyword 161, 162
 SetFrameConditions, [Setup] keyword 123
 [Setup]
 compile WinHelp 248
 conversion-template settings 81, 864
 convert generated files for MIF output 1008
 convert system variables to text 157
 convert TOC and IX 124
 for Help systems 205
 for MIF output 1008
 equations 137, 884
 exclude generated files 125
 export options and settings 84
 file names 120
 graphics
 export 726
 master- and reference-page 885
 output 130, 884, 885
 manage MIF files 111

MIF output 1008
 options determined at run time, *listed* 863
 process graphics 128, 130, 133, 884, 885
 master- and reference-page 885
 set-up options and settings 81
 subject to configuration overrides 925
 template settings 864, 865
 version numbers 1034
 WordPerfect settings 84
 SHGap, [WordOptions] keyword 153
 ShiftWideTablesLeft
 [Table] keyword 185
 [Tables] keyword 261
 ShipPath, [Automation] keyword 975
 activated by WrapAndShip 956
 ShowLog, [Logging] keyword 115
 SHSpannerAnchors, [WordOptions] keyword 160
 SHVertAdjust, [WordOptions] keyword 160
 SHWidth, [WordOptions] keyword 153
 Sideheads
 [HelpOptions] keyword 252
 [WordOptions] keyword
 convert sidehead formats 159
 set-up option 146
 SingleFlow, [WordOptions] keyword 157
 Slide, [HelpStyles] format property 268, 270
 SlideEnd, [Inserts] WinHelp keyword 822
 SlideStart, [Inserts] WinHelp keyword 822
 SmallCaps
 [HelpOptions] keyword 254
 [WordOptions] keyword 172
 SpaceAfterUnicode, [Defaults] keyword 148
 SpacelessMatch, [Options] keyword 104, 113
 SpKey, [HelpStyles] format property 269, 270, 286
 Start, [HelpBrowse] keyword 292
 Step, [HelpBrowse] keyword 292
 StretchMode, [Graphics] keyword 899
 StrippedCellPar, [Table] keyword 263
 StripTables, [Table] keyword 262
 StripTables, [Tables] keyword 267
 [StyleCodeStore], assign macro variable to paragraph format 804
 [StyleReplacements], merge formats 159
 for running headers and footers 156
 [Styles], map paragraph formats to Word styles 158

Suffix, [HelpStyles] format property 270, 282
 SuppressGTUnderlines, [Graphics]
 keyword 903
 SystemCommandWindow, [Automation]
 keyword 939
 SystemEndCommand, [Automation] keyword 938
 SystemStartCommand, [Automation]
 keyword 938
 SystemWrapCommand, [Automation] keyword 938

T

TableContinued, [Tables] keyword 184
 TableContVar, [Table] keyword 185
 TableFill
 [Table] keyword 185
 [Tables] keyword 262
 TableGraphics
 [Tables] keyword 262
 [WordOptions] keyword 185
 TableRules
 [Table] keyword 185
 [Tables] keyword 262
 TableSheet, [Table] keyword 185
 TableSheetVar, [Table] keyword 185
 TableTitles
 [Table] keyword 184, 261
 TableWidthsFixed, [Table] keyword 261
 TblColWidAdd, [Table] keyword 262
 TblColWidPct, [Table] keyword 262
 TblFullWidth, [Table] keyword 262
 Template, [WordOptions] keyword 149
 TemplateAutoUpdate, [WordOptions]
 keyword 149
 TemplateFileName, [Setup] keyword 864
 for chapter-specific templates 865
 set-up option 81
 [Templates] 859
 for document-specific settings 854
 for general configuration settings 851
 for macro libraries 851
 TextColor
 [HelpOptions] keyword 258
 [WordOptions] keyword 172
 [TextFlows] 113, 156
 TextScale, [Graphics] keyword 903

TextVertAdjust, [Graphics] keyword 903
 TextWidth, [Graphics] keyword 903
 TitleIndent, [HelpOptions] keyword 272
 TitleInRow, [Table] keyword 185
 TitleScroll, [HelpOptions] keyword 271
 TitleSpace, [HelpOptions] keyword 272
 TitleSuf, [HelpStyles] format property 270
 Top, [Inserts] Word keyword 822
 Topic, [HelpStyles] format property 269, 270
 for pop-up topics 268
 TopicEnd, [Inserts] WinHelp keyword 822
 TopicStart, [Inserts] WinHelp keyword 822
 TrailingTabs, [WordOptions] keyword 165
 Transparent, [Graphics] keyword 903

U

Uline, [HelpStyles] format property 270, 281
 UnderlineTabs, [WordOptions] keyword 165
 UseDefaultGraphicFormat, [Graphics]
 keyword 901
 UseDoneDialog, [Setup] keyword 113
 UseExistingDCL, [Setup] keyword 111
 determined at run time 863
 export embedded graphics 131
 export option 84
 UseExistingMIF, [Setup] keyword 111
 determined at run time 863
 export option 84
 UseFileIDs
 [HelpOptions] keyword 120, 273
 [WordOptions] keyword 120, 179
 UseFrame8MIF, [Setup] keyword
 for MIF output 1008
 UseFrame9MIF, [Setup] keyword
 for MIF output 1008
 UseFrameGenFiles, [Setup] keyword 125
 for MIF output 1008
 set-up option 81
 UseFrameImage, [Graphics] keyword 876
 UseFrameIX, [Setup] keyword 124
 for Help systems 205
 for MIF output 1008
 set-up option 81
 UseFrameTOC, [Setup] keyword 124
 for Help systems 205

 for MIF output 1007
 set-up option 81
 UseGraphicFileID, [Setup] keyword 133, 885
 FrameMaker export filters 726
 UseGraphicPreviews, [Graphics] keyword 130,
 884
 determined at run time 863
 export option 84
 turn off for native graphics export 128
 turn off for replaced graphics 131
 UseGreen, [HelpOptions] keyword 275
 UseHyperColor, [HelpOptions] keyword 139,
 281
 UseHyperlinks
 [HelpOptions] keyword 277
 [WordOptions] keyword 178, 182
 UseInitDialog, [Setup] keyword 113
 UseLocalFileID, [Setup] keyword 120, 122, 179,
 273
 UseLog, [Logging] keyword 115
 UseParaAnchors, RTF [WordOptions]
 keyword 172
 [UserVarPrompts], user variables 942
 [UserVars], user variables 941
 create a macro variable 796
 UseTextFrames, [WordOptions] keyword 152
 UseTopSpaceAbove, [Graphics] keyword 903

V

No entries for this letter

W

WhileMax, [Macros] keyword 816
 WildcardMatch, [Options] keyword 113
 Window, [HelpStyles] format property 270, 277
 WinHelpDocName, [Setup] keyword
 set-up option 60
 Word2000, [WordOptions] keyword 150
 Word2002, [WordOptions] keyword 150
 Word2003, [WordOptions] keyword 150
 correct graphics scale 192
 Word2007, [WordOptions] keyword 150
 Word2009, [WordOptions] keyword 150
 Word2010, [WordOptions] keyword 150

Word8, [WordOptions] keyword 149
 preserve graphics scale 192

[WordCntStyles], *deprecated*
 subject to configuration overrides 928

183

WordDocName, [Setup] keyword
 set-up option 60

[WordOptions]
 cross references 175, 177, 179, 180, 259
 equations 137
 fonts 170, 172
 footnotes 154
 for special characters 172
 for tabs 165
 formats 159, 160, 163
 graphics 152, 191, 874, 877, 895
 headers and footers 155, 156
 index 196
 line spacing 170
 ObjectIDs 175, 249
 page layout 152, 153
 reference frames 162
 set-up options and settings 146
 spacing 152
 special characters 172
 subject to configuration overrides 925
 tables 184, 185
 tabs 165
 templates 149
 text 157, 171, 172, 173
 WordPerfect 154

WordPerfect, [WordOptions] keyword 147

[WordReplacements] 174
 subject to configuration overrides 928

[WordSectionFiles], autonumbers 181

[WordStyles], print RTF format properties
 hide content 173
 make text an active link 182
 omit content 174
 replace content 174
 retain ObjectIDs 183
 subject to configuration overrides 928

[WordXrefFiles], cross references 181

WrapAllFrames, [Graphics] keyword 886

WrapAndShip, [Automation] keyword 956
 determined at run time 863
 export option 84

WrapAroundHFFrames, [WordOptions]
 keyword 155

WrapAroundTextFrames, [WordOptions]
 keyword 152, 191

WrapCopyFiles, [Automation] keyword 962
 activated by CompileHelp 972

WrapPath, [Automation] keyword 961
 activated by CompileHelp 972
 activated by WrapAndShip 956
 for WinHelp 248, 250, 334

WrapXrefs, [WordOptions] keyword 177

WriteAllGraphics, [Setup] keyword 130, 884
 determined at run time 863
 export option 84
 native graphics processing 128
 third-party graphics tools 131

WriteAllVarForms, [WordOptions] keyword 163

WriteAnums
 [HelpOptions] keyword 257
 [WordOptions] keyword 161, 162

WriteEquations, [Setup] keyword
 determined at run time 863
 does not affect equations 136
 export embedded graphics 131
 export option 84
 turn off for FrameMaker export 884
 use native graphics processing 128

WriteHelpProjectFile, [HelpOptions]
 keyword 246

WriteMasterPageGraphics, [Setup]
 keyword 885

WriteMissingForms, [WordOptions]
 keyword 156

WriteRefPageGraphics, [Setup] keyword 885

X

XrefFileDefault, [HelpOptions] keyword 260

XrefFileSuffix
 [HelpOptions] keyword 260
 [WordOptions] keyword 180

XrefHyper, [WordOptions] keyword 175

XrefLenLimit, [HelpOptions] keyword 261

Xrefs
 [HelpOptions] keyword 259
 [WordOptions] keyword 175

[XrefStyles], cross-reference formats 177, 260
 subject to configuration overrides 928

XrefType, [WordOptions] keyword 177, 259

XScroll, [HelpStyles] format property 270

Y

Y

No entries for this letter

Z

No entries for this letter

HTML/XML keyword index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

- Abbr, [HTMLParaStyles] WAI format
 property 768
 assign a value in [StyleCellAbbr] 769
 cell content abbreviation 768
- AbbrVal, [HTMLParaStyles] WAI format
 property 772
- AccessMethod, [Table] keyword 764
 apply the id/headers method to all tables 765
 apply the scope method to all tables 764
 avoid redundant attribute assignments 456
 effects on ColGroup property 784
 effects on RowGroup property 785
 overriding attributes 750
 use the scope method to identify cells 775
 WAI strategy for row/column markup 763
 ways to override 765
- AddCntWindowName, [HelpContents]
 keyword 291
- address, [ParaTags] format property 647
- AliasPrefix
 [MSHtmlHelpOptions] keyword 330
 [OmniHelpOptions] keyword 364
- AliasTitle, [MSHtmlHelpOptions] keyword 331
- AlignAttributes, [HTMLOptions] keyword
 CSS-dependent default value 688
 override paragraph properties 656
 XML default value 459
- ALink
 [HTMLParaStyles] format property
 create HTML Help ALink buttons 312
 for HTML-based Help 222
 for Oracle Help 400
 [MarkerTypes] property 221, 839
- ALink*, [MSHtmlHelpOptions] keywords:
 ALinkButtonGraphic 310
 ALinkButtonHeight 310
 ALinkButtonIcon 310
 ALinkButtonText 310
 ALinkButtonWidth 310
 ALinkEmptyTopic 310
 ALinkFlags 310
 ALinkText 310
 ALinkTextFont 310
 ALinkType 310
- ALinkRefs, [OmniHelpOptions] keyword 360
ALinkRefs, [OracleHelpOptions] keyword 400
- AllowEmptyAlt, [Graphics] keyword 718
- AllowNobr, [HTMLOptions] keyword 456
- AllowOverrides, [HTMLOptions] keyword
 CSS-independent default value 688
 override paragraph properties 657, 668
 XML default value 459
- AllowPartAppendix, [DITABookmapOptions]
 keyword 550
- AllowTbSplit, [Table] keyword
 convert tables to paragraphs 753
 designate split points 587
- AllowTbTitle, [Table] keyword
 convert tables to paragraphs 753
 titles for split files 595
- Alt, [HTMLParaStyles] format property for alt
 attribute 757
- AlwaysNestLists, [CSS] keyword 677
- Ansi, [HTMLOptions] keyword 431
- ANSI, [MarkerTypes] property 839
- Anum, [HTMLParaStyles] format property 677
- Anum, [HTMLParaStyles] format property, retain
 autonumbers 649
- [AnumCodeAfter], code after paragraph autonumber
 indent list items 678
 placement properties 823
 subject to configuration overrides 926
- [AnumCodeBefore], code before paragraph auto-
 number
 placement properties 823
 subject to configuration overrides 926
- AnumTabs, [HtmlOptions] keyword 649
- AppliedTemplateFlags, [Setup] keyword
 change template options 866
 set-up option 81
 template settings 864
- ApplyTemplateFile, [Setup] keyword 864
 change template options 866
 set-up option 81

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Archive*, [Automation] keywords:

- ArchiveCommand
 - activated by WrapAndShip 956
 - archive deliverables 973
 - JavaHelp set-up option 375
 - place deliverables 976
- ArchiveEndParams 974
- ArchiveExt 974
- ArchiveName 974
- ArchiveStartParams 973
- ArchiveVer 974

AskForUserVars, [Automation] keyword 942

ATagElement, [HTMLOptions] keyword 467

ATagLineBreak, [HTMLOptions] keyword 437

AttributeMarkers], map attributes to markers 135

[Attributes]

- for background images 725
- for <body> element 436
- for links 610
- for tables 736
- overridden by [TableAttributes] 736
- subject to configuration overrides 925
- when not to use 740

[Automation]

- default values in local_omsys.ini 59
- export options and settings 84
- options determined at run time, *listed* 863
- pre- and post-conversion system code 938
- produce deliverables 956
- user variables 942

Axis, [HTMLParaStyles] WAI format property 768

AxisVal, [HTMLParaStyles] WAI format property 772

B

[Base], default font and size 664

- subject to configuration overrides 925

Basefont, [HTMLOptions] keyword 664, 688, 701

BeginFile, [Inserts] keyword 429

BinaryIndex

- [MSHtmlHelpOptions] keyword 320

BinaryTOC

- [MSHtmlHelpOptions] keyword 320

Blockquote, [ParaTags] format property 647

BodyContentOnly, [HTMLOptions] keyword 450

Bold, [HTMLParaStyles] or [HTMLCharStyles]
format property 657

BookFileName, [DocBookOptions] keyword 562

BookFileName, [DocBookOptions] keyword 562

BookLibrary, [DITABookmapOptions]
keyword 548

BookMapID, [DITAOptions] keyword 543

BookMapName, [DITAOptions] keyword 541

BookMapTitle, [DITAOptions] keyword 542

BookmapType, [DITABookmapOptions]
keyword 548

BookMeta, [DITABookmapOptions] keyword 549

BookSubtitle, [DITABookmapOptions]
keyword 548

BookTitle, [DITABookmapOptions] keyword 548

Border, [Table] keyword 740

- overridden by [TableAttributes] 736, 740
- set-up option 426

Bottom, [Inserts] keyword 599, 600, 821

- position a navigation macro 642

C

CaselessMatch, [Options] keyword 113

- case sensitivity of FileIDs 121

CaseSensitiveIndexCompare, [Index]
keyword 218

CellAlignAttributes, [Table] keyword 739

- XML default value 459, 464

CellAttribute, [HTMLParaStyles] format
property 738

CellAttribute, [HTMLParaStyles] format
property 768

CellAttribute, [HTMLParaStyles] WAI format
property 768

CellColorAttributes, [Table] keyword 739

- XML default value 459, 464

Center, [HTMLParaStyles] format property 656

CGElems, [TableAccess] property 733

ChangeFileNameSpaces, [HTMLOptions]
keyword 949

[CharacterRangeClasses], assign classes to Uni-
code character ranges 695

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- [CharClasses]
 - default use for CSS class names 682
 - map character formats for XML 463
 - map character formats to span classes 694
 - subject to configuration overrides 925
- [CharConvert], map special characters 660
 - for JavaHelp 384
- [CharStyle*] sections
 - [CharStyleCode*] sections
 - all subject to configuration overrides* 926
 - [CharStyleCodeAfter] 823
 - [CharStyleCodeBefore] 823
 - [CharStyleCodeEnd] 823
 - [CharStyleCodeReplace] 823
 - [CharStyleCodeStart] 823
 - [CharStyleCSS] 700
 - [CharStyleLinkSrc] 823
- [CharTags]
 - assign HTML tags to character formats 653
 - assign XML tags to character formats 463
 - map character formats to CSS span classes 694
 - subject to configuration overrides 927
 - tags used for CSS classes by default 682
- CheckAllRefs, [HTMLOptions] keyword 618
- CheckLinkLog, [Setup] keyword: log broken links 112, 428
- CheckLinks, [Setup] keyword: check for broken links 112, 428
- [ChmFiles], map source files to .chm files 307, 337
- ChmFormat, [MSHtmlHelpOptions] keyword 308
- ClassIsTag, [CSS] keyword 692
 - map CSS class names to XML tags 463
 - use tag names for CSS class names 696
 - XML default value 459
- ClassSpaceChar, [HtmlOptions] keyword 691
- ClickBlockToClose, [DropDowns] keyword 233
- CloseFigAfterImage, [DITAOptions] keyword 517
- CloseFigAfterImage, [DocBookOptions] keyword 581
- CloseOldWindow, [OmniHelpOptions] keyword 352
- CloseStrippedTables, [Table] keyword 515
- Code, [MarkerTypes] property 839
- Code*, [HTMLCharStyles] format properties
 - CodeAfter 823
 - CodeBefore 823
 - CodeEnd 823
 - CodeReplace 823
 - CodeStart 823
- Code*, [HTMLParaStyles] format properties
 - CodeAfter 466, 804, 823
 - CodeAfterAnum 823
 - CodeBefore 466, 804, 823
 - CodeBeforeAnum 823
 - CodeEnd 647, 804, 823
 - CodeReplace 312, 325, 443, 823
 - CodeStart 647, 804, 823
 - CodeStore 804
 - capture FrameMaker autonumbers 944
 - create a macro variable 796
 - difference from TextStore 805
- ColGroup, [HTMLParaStyles] WAI format property 768
 - in [Table]ColGroupHead cells 778
 - use header cells to define column groups 766
- ColGroupElements, [Table] keyword 732
 - apply scope method to all tables 764
 - for browser-dependent table tags 731
 - override column group settings 733
- ColGroupHead, [Table] WAI keyword 778
 - column-group extent 779
 - id/header table cell attribute 778
- ColGroupIDs, [Table] WAI keyword 778
 - column-group extent 779
 - id/header table cell attribute 778
 - override for selected tables 784
 - set by AccessMethod=IDheaders 765, 767
- ColHead, [Table] WAI keyword 782
 - column extent 783
 - id/header table cell attribute 778
- ColIDs, [Table] WAI keyword 782
 - column extent 783
 - id/header table cell attribute 778
 - override for selected tables 784
- Color*, [HTMLParaStyles] or [HTMLCharStyles] format property 657, 669
- [Colors]
 - correct CMYK colors 439
 - map color names to values 439
 - mappings affect CSS 682
 - override for paragraph formats 657
 - table row color 746
- [Colors], specify text colors 669

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- ColSpanHead, [Table] WAI keyword 780
 - column-span extent 781
 - id/header table cell attribute 778
- ColSpanIDs, [Table] WAI keyword 780
 - column-span extent 781
 - dependencies 765
 - id/header table cell attribute 778
 - override for selected tables 784
- CombineIndexLevels, [Index] keyword 214
- Comment
 - [HTMLOptions]GeneratorTag setting 433
 - [HTMLParaStyles] format property for scripts 650
 - [HTMLParaStyles] or [HTMLCharStyles] format property 650
- CompileHelp, [Automation] keyword
 - compile HTML Help project 333, 971
 - determined at run time 863
 - export option 84
 - set up HTML Help project 299
- Compiler, [MSHtmlHelpOptions] keyword 334
- CompoundWordChars, [OmniHelpOptions] keyword 362
- [ConditionAttributes]
 - convert to DITA element attributes 534
 - convert to HTML/XHTML element attributes 447
- ConditionCharTag, [HTMLOptions] keyword 446, 533
- [ConditionOptions], display FrameMaker conditions via CSS 447
- [ConditionsShown], apply FrameMaker conditions 123
- Config, override configuration settings
 - [HTMLParaStyles] format property 931
 - [MarkerTypes] property 839
- Configs, [Templates] keyword 851, 859, 862
 - precedence of settings 919
- Confluence, [HTMLOptions] keyword 449
- ConfluenceLinkEnd, [HTMLOptions] keyword 450
- ConfluenceLinkPage, [HTMLOptions] keyword 450
- ConfluenceLinkPageEnd, [HTMLOptions] keyword 450
- ConfluenceLinks, [HTMLOptions] keyword 450
- ConfluenceLinkStart, [HTMLOptions] keyword 450
- ConfluenceLinkText, [HTMLOptions] keyword 450
- ConfluenceLinkTextEnd, [HTMLOptions] keyword 450
- ContentModel, [DocBookOptions] keyword 561
- ContentModel, DocBookOptions] keyword 908
- Contents
 - [HTMLParaStyles] format property
 - HTML-based Help contents entries 210
 - [HtmlStyles] format property
 - retain ObjectIDs 445, 621
 - [JavaHelpOptions] ListType value 207, 320, 375
 - [MSHtmlHelpOptions] ListType value 207, 299, 320
- ContentsLocalValuePrefix,
 - [MSHtmlHelpOptions] keyword 338
- ContentsNamesFileOnly, [MSHtmlHelpOptions] keyword 323
- ContentType, [HTMLOptions] keyword 435, 461
- ContextAnchors, [EclipseHelpOptions] keyword 411, 418
- ContextDescription, [EclipseHelpOptions] keyword 411, 419
- ContextFileName, [EclipseHelpOptions] keyword 418
- ContextID, [EclipseHelpOptions] keyword 418
- ContextPluginName, [EclipseHelpOptions] keyword 418
- ConversionDPI, [HTMLOptions] keyword 436, 721
- ConvertVariables, [Setup] keyword 437
 - convert system variables to text 114
 - set-up option 81
- CopyAfterFiles, [Automation] keyword 964
- CopyAfterFrom, [Automation] keyword 964
- CopyBeforeFiles, [Automation] keyword 960
- CopyBeforeFrom, [Automation] keyword 960
- CopyCssFrom, [Automation] keyword 969
 - activated by CompileHelp or FTSCCommand 972
 - activated by WrapAndShip 956
- CopyGraphicsFrom, [Automation] keyword 966
 - activated by CompileHelp or FTSCCommand 972

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- activated by `WrapAndShip` 956
- locate graphics for HTML Help 302
- locate graphics for OmniHelp 349
- `CopyOriginalGraphics`, [Automation]
 - keyword 965
- `CshMapFile`, [MSHtmlHelpOptions] keyword
 - set-up option 299
 - use symbolic IDs for CSH links 329
- `CshMapFileNumIncrement`,
 - [MSHtmlHelpOptions] keyword 329
- `CshMapFileNumStart`, [MSHtmlHelpOptions]
 - keyword 329
- [CSS] 682–701
 - file options 684
 - for XML 463
 - link options 611
 - list attributes 678
- Css*, [CSS] keywords:
 - `CssBodyFontSize` 698
 - `CssBodyFontTag` 699
 - `CssBodyFontUnit` 698
 - `CssBrowserDetect` 684
 - `CssFileName` 684
 - name CSS files 686
 - set-up option 426, 479, 561
 - `CssFontUnitDec` 699
 - `CssFontUnits` 699
 - `CssIndentBaseSize` 699
 - `CssIndentBaseUnit` 699
 - `CssIndentUnitDec` 699
 - `CssIndentUnits` 699
 - `CSSLinkNS4` 691
 - `CssPath` 686
 - destination for `CssCopyFiles` 964
 - place CSS files for assembly 969
- `CssCopyFiles`, [Automation] keyword 969
 - activated by `CompileHelp` or `FTSCommand` 972
- [`CSSEndMacro`], ending code for CSS file 700
- `CSSReplace`, [HTMLParaStyles] or
 - [HTMLCharStyles] format property 700
- [`CSSStartMacro`], starting code for CSS file 700
 - specify default font size 699
 - when CSS is generated each time 607
- `CtrlCssName`, [OmniHelpOptions] keyword 352

D

- Default, [*JavaHelp* window] parameter 394
- `DefaultChmFile`, [MSHtmlHelpOptions]
 - keyword 301
 - map CHM files 336
 - set-up option 299
 - syntax for inter-CHM-file links 308
- `DefaultTarget`, [HTMLOptions] keyword 451, 725
- `DefaultTopic`, [JavaHelpOptions] keyword 382
 - set-up option 376
- `DefaultTopic`, [OracleHelpOptions]
 - keyword 382
- `DefaultTopicFile`
 - [MSHtmlHelpOptions] keyword 301
 - set-up option 299
 - [OmniHelpOptions] keyword 348
 - set-up option 347
- `DefCharElem`, [DITAOptions] keyword 494
- `DefCharElem`, [DocBookOptions] keyword 569
- `DefParaElem`, [DITAOptions] keyword 489
- `DefParaElem`, [DocBookOptions] keyword 566
- `DefTableElem`, [DITAOptions] keyword 511
- `DefTopic`, [DITAOptions] keyword 525
- Delete
 - [HTMLCharStyles] format property, override placement 926
 - [HTMLParaStyles] format property 772, 950
 - delimit extracts 593
 - do not use for `CodeStore` paragraphs 804
 - eliminate glossary entries from *JavaHelp* TOC 393
 - eliminate unwanted paragraphs 653
 - enable/disable extract processing 591
 - for configuration overrides 931
 - hide `TextStore` paragraphs 803
 - hide WAI markup 756, 757, 758
 - HTML Help TOC-only entries 322
 - name split files 950
 - override placement 926
 - paragraph formats for `<meta>` tags 435
 - prevent duplicate file names 952
 - [HtmlStyles] format property
 - CSH paragraphs for HTML Help 328
 - eliminate invisible paragraphs 652
 - eliminate page numbers from generated lists 445

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- suppress page numbers in HTML Help [325](#)
 - [MarkerTypes] property [839](#)
 - hide **Index** markers from JavaHelp [386](#)
 - must be specified last [840](#)
 - suppress markers [841](#)
- [XrefStyles] format property [618](#)
- DeleteExistingDCL, [Setup] keyword [112](#)
- DeleteExistingMIF, [Automation] keyword
 - activated by CompileHelp or FTSCOMMAND [972](#)
- DeleteExistingMIF, [Setup] keyword [111](#)
- DescriptionIsFirstLabel,
 - [EclipseHelpOptions] keyword [411](#), [418](#)
- [DITAAliases], alternate names for a format [490](#)
- [DITABookmapFiles], roles for bookmap components [551](#)
- [DITABookmapHrefFormats], format attribute values for wrapper elements [556](#)
- [DITABookmapHrefs], href attribute values for wrapper elements [556](#)
- [DITABookmapHrefScopes], scope attribute values for wrapper elements [556](#)
- [DITABookmapHrefTypes], type attribute values for wrapper elements [556](#)
- [DITABookmapOptions], configure <bookmap> element [548](#)
- [DITABookmapOutputclasses], outputclass attribute values for wrapper elements [556](#)
- [DITABookmapTitles], <navtitle> values for wrapper elements [556](#)
- [DITACHarAttributes], assign attributes to inline elements [498](#)
- [DITACHarTags]
 - character formats to DITA elements [492](#)
- [DITACHarTypographics], multiple typographic elements [494](#)
- [DITACloseAfter], close parent element after current block [507](#)
- [DITACloseBefore], close ancestor element(s) before current block [506](#)
- [DITAContentModels], specialize DITA topic type [908](#), [914](#)
- [DITAElementSets], specify alternate ancestors [504](#)
- [DITAFirst], specify first-child status [505](#)
- [DITAImageParents], parents for image or figure element [516](#)
- [DITALevels], element levels [510](#)
- [DITAMapLevels], topic levels in maps [544](#)
- [DITAMapUsage], topic roles maps [545](#)
- [DITAOpenAfter], open new element(s) after current block [507](#)
- [DITAOpenBefore], open ancestor element(s) before current block [507](#)
- [DITAOptions], options for DITA XML output [480](#)
- [DITAParaAttributes], assign attributes to block elements [497](#)
- [DITAParaTags]
 - alternate paragraph formats to DITA elements [491](#), [527](#)
 - paragraph formats to DITA elements [487](#)
- [DITAParaTypographics], multiple typographic elements [494](#)
- [DITAParentAttributes], assign attributes to interpolated parents [498](#)
- [DITAParents], possible parents for elements [502](#)
- [DITAPreformatted], preserve whitespace in block elements [499](#)
- [DITARelBookGroups], collection-type attributes for topic types [547](#)
- [DITARelGroups], topic collection-type [547](#)
- [DITATableAttributes], assign attributes to table types [511](#)
- [DITATableParents], parents for root table element [512](#)
- [DITATables], map formats to table types [511](#)
- [DITATopicFileNamePrefix], prefix split-file names by topic type [521](#)
- DITATopicIDLowerCase, [DITAOptions] keyword [526](#)
- DITATopicIDSpaceChar, [DITAOptions] keyword [526](#)
- DITATopicIDUnderscore, [DITAOptions] keyword [526](#)
- [DITATopicRootAttrs], assign attributes to the root element of a topic [497](#)
- DITAVer, [DITAOptions] keyword [480](#)
- default for FrameMaker 8 import [481](#)
- DListDD, [HTMLParaStyles] format property [675](#), [677](#)
- [DocBookAlias], alternate names for a format [567](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- [DocBookCharAttributes], assign attributes to inline elements [572](#)
- [DocBookCharTags]
 - character formats to DocBook elements [568](#)
- [DocBookCloseAfter], close parent element after current block [578](#)
- [DocBookCloseBefore], close ancestor element(s) before current block [577](#)
- [DocBookElementSets], specify alternate ancestors [575](#)
- [DocBookFirst], specify first-child status [576](#)
- [DocBookImageParents], parents for image element [582](#)
- [DocBookLevels], element levels [580](#)
- [DocBookOpenAfter], open new element(s) after current block [578](#)
- [DocBookOpenBefore], open ancestor element(s) before current block [578](#)
- [DocBookOptions] [561](#)
- [DocBookParaAttributes], assign attributes to block elements [571](#)
- [DocBookParaIDs]
 - include ID attributes in block elements [570](#)
- [DocBookParaTags]
 - paragraph formats to DocBook elements [565](#)
- [DocBookParentAttributes], assign attributes to interpolated parents [572](#)
- [DocBookParentIDs]
 - include ID attributes in interpolated parent elements [570](#)
- [DocBookParents], possible parents for elements [574](#)
- DocBookRoot, [DocBookOptions] keyword [562](#)
- [DocBookTableParents], parents for root table element [581](#)
- Document, [Templates] keyword [854](#), [859](#), [860](#)
- Drop*, [DropDowns] keywords
 - DropBlockEnd [233](#)
 - DropBlockStart [233](#)
 - DropButton [233](#)
 - DropButtonAttr [233](#)
 - DropButtonCloseLabel [232](#)
 - DropButtonOpenLabel [232](#)
 - DropClass [233](#)
 - DropCloseIcon [233](#)
 - DropCloseIconAlt [231](#)
 - DropCloseIconFile [231](#)
 - DropDivAttr [233](#)
 - DropDownBlock [228](#)
 - DropIDPrefix [232](#)
 - DropJSCode [234](#)
 - DropJSLocation [234](#)
 - DropLinkAttr [232](#)
 - DropLinkEnd [232](#)
 - DropLinkPara [233](#)
 - DropLinkParaEnd [233](#)
 - DropLinkParaStart [233](#)
 - DropLinkParaText [233](#)
 - DropLinkStart [232](#)
 - DropLinkType [230](#)
 - DropOpenIcon [233](#)
 - DropOpenIconAlt [231](#)
 - DropOpenIconFile [231](#)
 - DropText [232](#)
- DropDown, [HTMLParaStyles] format property [228](#)
- DropDownEnd, [HTMLParaStyles] format property [228](#)
- DropDownLink, [HTMLParaStyles] format property [228](#)
- DropDownLink, [HtmlParaStyles] or [HtmlCharStyles] format property [228](#)
- [DropDowns], create expandable drop-down sections [230](#)
- DropDownStart, [HtmlCharStyles] format property [228](#)
- DropDownStart, [HTMLParaStyles] format property [228](#)
- DropInvalidParaTag, [DITAOptions] keyword [490](#)
- DropInvalidParaTag, [DocBookOptions] keyword [566](#)
- DumpToFile, DITA [Topic] content-model keyword [918](#)

E

- [EclipseHelpOptions], set-up options and settings [405](#)
- EclipseVer, [EclipseHelpOptions] keyword [406](#)
- EditorFileName, [Logging] keyword [115](#)
- [ElementSets], content-model section; define sets

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- of DITA elements 910
- EmptyGraphPath, [Automation] keyword 967
 - activated by CompileHelp or FTSCCommand 972
 - when effective 962
- EmptyJavaGraphSubdir, [JavaHelpOptions] keyword 379
- EmptyJavaGraphSubdir, [OracleHelpOptions] keyword 379
- EmptyJavaHTMLSubdir, [JavaHelpOptions] keyword 379
- EmptyJavaHTMLSubdir, [OracleHelpOptions] keyword 379
- EmptyOutputDir, [Automation] keyword 958
 - activated by CompileHelp or FTSCCommand 972
 - dependencies 959
 - when effective 958
- EmptyOutputFiles, [Automation] keyword 958
 - activated by CompileHelp or FTSCCommand 972
 - when to include 959
- EmptyParaContent, [HTMLOptions] keyword 652
- EmptyTbCellContent, [Table] keyword 744
 - eliminate default content for XML 464
- EmptyWrapPath, [Automation] keyword 962
 - activated by CompileHelp or FTSCCommand 972
 - dependencies 968
- Encoding, [HTMLOptions] keyword 432
 - for Eclipse Help 412
 - for XML 460
 - prevent character mapping 663
- End, [Inserts] keyword 599, 600, 821
 - for framesets 451
- [End], dummy section to end settings 107
- [End], dummy section to replace
 - [MacroVariables] 788
- EndingFSButton, [NavigationMacros] keyword 641
- EndingNextFSButton, [NavigationMacros] keyword 641
- EndingNextFSMacro, [NavigationMacros] keyword 640, 828
 - scope 640
- Entities, [Inserts] keyword 429, 821
- EqSuffix, [HTMLOptions] keyword 137
- EquationExportDPI, [Setup] keyword 137
 - convert equations 726
 - graphic output format and resolution 884
- EquationFrameExpand, [Setup] keyword 137, 726
- Export*, [GraphExport] keywords:
 - ExportBmpFiles 881
 - ExportCdrFiles 881
 - ExportEpsFiles 881
 - ExportGifFiles 881
 - ExportJpgFiles 881
 - ExportNameChars 134
 - ExportNumDigits 134
 - ExportPctFiles 881
 - ExportPcxFiles 881
 - ExportPngFiles 881
 - ExportRfFiles 881
 - ExportTifFiles 881
 - ExportWmfFiles 881
 - ExportWpgFiles 881
- Extr*
 - [Graphics] extract keywords:
 - ExtrGraphClass 606
 - ExtrGraphHigh 605
 - ExtrGraphSuffix 601, 603, 605
 - ExtrGraphTarget 606
 - ExtrGraphThumbnail 605
 - ExtrGraphWide 606
 - [HTMLParaStyles] extract format properties
 - ExtrDisable 592
 - ExtrEnable 592
 - ExtrEnd 592
 - ExtrFinish 592
 - ExtrStart 592
 - [Inserts] keywords:
 - ExtrBottom 600
 - ExtrHead 600
 - ExtrHeadEnd 600
 - ExtrTop 600
 - [MarkerTypes] properties
 - ExtrBottom 839
 - ExtrDisable 839
 - ExtrEnable 839
 - ExtrEnd 839
 - ExtrFinish 839
 - ExtrHead 839
 - ExtrReplace 839
 - ExtrStart 839
 - ExtrTop 839
- [Extr*], extract-file sections
 - all subject to configuration overrides 927*
 - [ExtrBottom] 600
 - [ExtrHead] 600

ABCDEFGHIJKLMNOPQRSTUVWXYZ

[ExtrReplace], extract replacement code 603
 [ExtrTitle] 602
 [ExtrTop] 600
 ExtractEnable, [HTMLOptions] keyword 591

F

Figure, [HTMLParaStyles] format property, ensure wrapping DITA image in <fig> 517
 FigureTitleStartsFigure, [DITAOptions] keyword 517
 FigureTitleStartsFigure, [DocBookOptions] keyword 581
 [FileIDs] 121
 deprecated for main configuration file 122
 mif2go.ini section 121
 FileName
 [HTMLParaStyles] format property 948
 [MarkerTypes] property 839
 importance of processing order 587, 840
 name split and extract files 947
 FileNameSpaceChar, [HTMLOptions] keyword 949
 FileSuffix, [Setup] keyword 110, 460
 for DITA output 480
 for DocBook output 561
 output file extension for ASCII DCL 1012
 First*, [Inserts] keywords:
 FirstBottom 600
 position local TOCs 634
 FirstEnd 600
 FirstFrames 600
 FirstHead 600
 FirstHeadEnd 600
 FirstTop 600
 FixGraphSpaces, [Graphics] keyword 889
 FixMacroQuotes, [Macros] keyword 790
 FM8Import, [DITAOptions] keyword 481
 Font, [Base] keyword 664
 [Fonts], remap fonts 664
 assign CSS generic font family 698
 effect on CSS rendition 682
 [FontSizes], map points to HTML sizes 665
 FootClass, [CSS] keyword 694
 FootInlineIDPrefix, [HTMLOptions] keyword 564, 672

FootInlineParaTag, [HTMLOptions] keyword 564, 672
 FootInlineRefTag, [HTMLOptions] keyword 564, 672
 FootInlineTag, [HTMLOptions] keyword 672
 FootnoteEndCode, [HTMLOptions] keyword 673
 Footnotes, [HTMLOptions] keyword 564, 672
 XML default value 459
 FootnoteSeparator, [HTMLOptions] keyword 672
 FootnoteStartCode, [HTMLOptions] keyword 673
 FootnoteWrapClass, [DITAOptions] keyword 528
 FootnoteXref, [DITAOptions] keyword 529
 FootTagLast, [Table] keyword 733
 ForceStartTopic, [DITAOptions] keyword 524
 FrameAbove, [HtmlStyles] format property 651
 FrameBelow, [HtmlStyles] format property 651
 FrameHigh, [OmniHelpOptions] keyword 353
 FrameOptions, [OmniHelpOptions] keyword 353
 Frames, [Inserts] keyword 599, 600, 821
 for framesets 451
 Frameset, [OmniHelpOptions] keyword 353
 FrameWide, [OmniHelpOptions] keyword 353
 FRowsN, [TableAccess] keyword 734, 735
 FTSCCommand, [JavaHelpOptions] keyword
 for JavaHelp 388
 set-up option 375
 FTSCCommand, [racleHelpOptions] keyword
 for Oracle Help 389

G

GenerateBook, [Setup] keyword 126
 set-up option 81
 template import problem 866
 GeneratorTag, [HTMLOptions] keyword 433
 [GlossFiles]
 for hover text 449
 GlossPrefix, [JavaHelpOptions] keyword 393
 GlossSpace, [JavaHelpOptions] keyword 393
 GlossSuffix, [JavaHelpOptions] keyword 393
 GlossTerm, [HTMLParaStyles] JavaHelp format

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- property 392
- GlossTitle, [HtmlStyles] format property, provide for hover text 448
- GlossTitlePath, [HTMLOptions] keyword 449
- [GlossTitles]
 - for hover text 448
- [GraphAlign] 714
 - left-align all images 704
 - subject to configuration overrides 930
- GraphAlignAttributes, [Graphics]
 - keyword 715
 - default depends on [CSS]UseCSS 688
- [GraphALT], image alt attribute
 - for image maps 723
- [GraphALT], image alt tags 719
 - subject to configuration overrides 930
- [GraphAttr], image attributes 718
 - eliminate blue borders 723
 - subject to configuration overrides 930
- GraphClass, [Graphics] keyword 463, 693
- GraphCopyFiles, [Automation] keyword 966
 - activated by CompileHelp or FTSCCommand 972
- [GraphDpi], image DPI 721
 - subject to configuration overrides 930
- [GraphEndMacros], code after images 711, 715
 - subject to configuration overrides 930
- [GraphExport]
 - export embedded graphics 1012
- [GraphExport], export embedded graphics 707, 880, 1012
 - from OLE objects 882
 - set-up options and settings 1011
 - subject to configuration overrides 925
- [GraphFiles], replace graphics 707, 888, 968
 - path overrides 888
 - subject to configuration overrides 930
- [GraphGroup], create graphics groups 710
 - assign with *Config marker 930
 - override with FrameMaker Object Attributes 896
 - subject to configuration overrides 930
- [GraphHigh], image height in pixels 720
 - property of extracted graphic 607
 - related to predefined macro variables 711
 - subject to configuration overrides 930
- GraphicExportDPI, [Setup] keyword 130, 721, 884
- GraphicExportFormat, [Setup] keyword 130, 884
 - for equations 725
- GraphicNameDigits, [Setup] keyword 134, 885
- [Graphics]
 - class name for anchor paragraph 463, 693
 - export options and settings 84
 - fix graphics file names 889
 - graphics location for JavaHelp 381
 - options determined at run time, *listed* 863
 - position graphics 714
 - relocate graphics files 704, 705, 887
 - remove path information 335
 - replace EPSI graphics 877
 - replace graphics 888, 968
 - subject to configuration overrides 925
 - third-party graphics tools 131
 - thumbnails for extract links 605
 - use existing graphics files 706, 884, 889
 - use title for alt 724
 - include or omit image attributes
 - for DITA XML 518
 - for DocBook XML 582
 - for generic XML 464
 - for HTML 720
- GraphicsFirst, [Setup] keyword
 - export embedded graphics 1012
 - export master- and reference-page graphics 885
 - process all graphics first 132
 - process embedded graphics separately 133
 - process only graphics 88
- [GraphIndents], indent images 717
 - subject to configuration overrides 930
 - unindent images 303
- [GraphParaAlign], position graphics 715
 - subject to configuration overrides 930
- GraphPath, [Graphics] keyword 705, 887, 968
 - for JavaHelp and Oracle Help 381
 - include unconverted referenced graphics 890
 - overridden by [GraphFiles] 888
 - overrides [GraphFiles] 888
 - replace EPS graphics 877
- GraphPathOverrides, [Graphics] keyword 888, 929, 968
- [GraphReplaceMacros], code instead of image 711
 - subject to configuration overrides 930

ABCDEFGHIJKLMNOPQRSTUVWXYZ

[GraphRightSpacers], indent images 717
subject to configuration overrides 930

GraphScale, [Graphics] keyword

eliminate attributes

for DITA XML 518

for DocBook XML 582

for generic XML 464

for HTML 720

eliminate attributes for HTML 704

XML default value 459

[GraphScale], scale images 720

related to predefined macro variables 711

subject to configuration overrides 930

[GraphStartMacros] 710

add space before a graphic 717

subject to configuration overrides 930

GraphSubdir, [JavaHelpOptions] keyword 379

GraphSubdir, [OracleHelpOptions]

keyword 379

GraphSuffix, [Graphics] keyword 888, 968

replace referenced graphics 706

third-party graphics tools 131

use referenced graphics without converting 706

[GraphSuffix], replace graphics file

extension 706, 888, 968

[GraphWide], width of image in pixels 720

property of extracted graphics 607

related to predefined macro variables 711

subject to configuration overrides 930

GraphWrapPara, [Graphics] keyword 464, 713

H

h1 - h6, [ParaTags] format properties 647

HColsN, [TableAccess] keyword 735

default header columns 734

effect on [Table]ScopeRow 776

Head, [Inserts] keyword 599, 821

customize CSS link tag 690

for CSS selection macro 685, 689

for HTML Help KeyHelp pop-up 306

for solitary file 600

HeadEnd, [Inserts] keyword 599

for solitary file 600

HeadFootBodyTags, [Table] keyword 732

choose a row-group method 780

default header/footer counts 734

enable [Table*Attributes] 751

identify table cells via scope 776

RowGroup property 785

with scope method 764, 767

Height, [JavaHelp window] parameter 394

Helen, [JavaHelpOptions] keyword 385

[HelpContentsLevels]

assign heading levels for split overrides 589

assign local TOC levels 632

check assigned split points 586

exclude links 445

for HTML-based Help 210

subject to configuration overrides 927

HelpFileLanguage, [MSHtmlHelpOptions]

keyword 332

HelpFileName

[JavaHelpOptions] keyword 382

set-up option 376

[MSHtmlHelpOptions] keyword 300

set-up option 299

[OmniHelpOptions] keyword 348

set-up option 347

[OracleHelpOptions] keyword 382

HelpMerge, [MarkerTypes] property 839

[HelpMerge], merge help projects

for HTML Help 339

for OmniHelp 367, 369

[HelpMergePaths], merge JavaHelp or Oracle Help

helpsets 400

HFBTags, [TableAccess] override 733, 751

HHCPProperties, [MSHtmlHelpOptions]

keyword 324

HHKProperties, [MSHtmlHelpOptions]

keyword 324

HHPFileName, [MSHtmlHelpOptions]

keyword 301

archive deliverables 975

set-up option 299

[HHWindows], secondary windows

for HTML Help 318

HideWhiteText, [HTMLOptions] keyword 652

HistoryFileName, [Logging] keyword 115

HrefAttribute, [HTMLOptions] keyword 467,
563

HRowsN, [TableAccess] keyword 735

default header row count 734

ABCDEFGHIJKLMNOPQRSTUVWXYZ

effect on [Table]ScopeCol 776

HSFileName, [JavaHelpOptions] keyword 391

- archive JavaHelp deliverables 975
- name JavaHelp helpset file 382
- set-up option 376

HSFileName, [OracleHelpOptions] keyword

- name Oracle Help helpset file 382

HSPathNames, [JavaHelpOptions] keyword 383

HSPathNames, [OracleHelpOptions]

- keyword 383

HTMConfig

- [HTMLParaStyles] format property 931
- [MarkerTypes] property 839

[HTMLCharStyles]

- subject to configuration overrides 927

HTMLComment, [MarkerTypes] property 839

HTMLDocName, [Setup] keyword

- set-up option 60

HTMLDocType, [HTMLOptions] keyword 429

- for framesets 451

HTMLDTD, [HTMLOptions] keyword 430

- for framesets 451

[HtmlFiles], rename split files 946

[HTMLOptions]

- for declarations 429, 430, 432, 435, 436, 451
- for extracts 591
- for footnotes 564, 672
- for framesets 451
- for graphics 713, 886
- for links 443, 451, 725
- for preformatted text 437, 670, 671
- for split files 587, 588, 590, 948
- for tables 747
- for Word cross references 114
- for XML 468
- subject to configuration overrides 925

[HTMLParaStyles]

- for extracts 592
- for HTML Help 222
- for images 757
- for links 758, 759
- for split files 595
- for WAI table attributes 768
- subject to configuration overrides 927

HTMLSubdir, [JavaHelpOptions] keyword 379, 381

HTMLSubdir, [OracleHelpOptions] keyword 379

HTMLVersion, [HTMLOptions] keyword 429

HyperSpaceChar, [HTMLOptions] keyword 613

I

IDAttrName, [HTMLOptions] keyword 135

IDFileName, [Setup] keyword 120, 623, 1027

- determined at run time 863

IDheaders, [Table]AccessMethod option 764

- apply id/headers to all tables 765
- ColGroup dependency 784
- RowGroup dependency 785

IDRefAttrName, [HTMLOptions] keyword 135

IDs, [TableAccess] property, 765

IdxButtons, [OmniHelpOptions] keyword 359

IdxExpand, [OmniHelpOptions] keyword 358

IdxFilename, [EclipseHelpOptions]

- keyword 410

IdxGroupsOpen, [OmniHelpOptions]

- keyword 358

IdxIcoBase, [OmniHelpOptions] keyword 359

IdxOpenLevel, [OmniHelpOptions] keyword 358

IECssName, [OmniHelpOptions] keyword 350

IECtrlCssName, [OmniHelpOptions]

- keyword 352

IgnoreCharsIX, [Index] keyword 217

IgnoreLeadingCharsIX, [Index] keyword 217

IgnoreWrap, [HTMLOptions] keyword 438, 670

Image, [JavaHelp window] parameter 394

ImageParents, [DITAOptions] keyword 516

ImageParents, [DocBookOptions] keyword 581

ImgSrcAttr, [HTMLOptions] XML keyword 468

ImgTagElement, [HTMLOptions] XML

- keyword 468

ImportDocProps, [Setup] keyword 865

ImportGraphics, [GraphExport] keyword, export

- embedded graphics 707, 880
- by image format 880
- OLE objects 881
- separately 133
- via ASCII DCL output 1012

IncludeVersionPI, [EclipseHelpOptions]

- keyword 409

Index

ABCDEFGHIJKLMNOPQRSTUVWXYZ

[JavaHelpOptions]ListType value 207, 320
 [MSHtmlHelpOptions]ListType value 207, 320
 [Index]
 configure Help index entries 213
 [IndexMarkerOutputClass], href outputclass for indexterms 554
 IndexRanges, [HtmlOptions] keyword 213
 IndexSortLocale, [HtmlOptions] keyword 218
 IndexSortType, [HtmlOptions] keyword 218
 IndexWrapClass, [DITAOptions] keyword 528
 [Inserts], insert code at predefined locations 599, 821
 subject to configuration overrides 925
 InternalTableCaption, [Table] keyword 563, 747
 Ital, [HTMLPararStyles] or [HTMLCharStyles] format property 657

J

JarCommand, [JavaHelpOptions] keyword 391
 JarCommand, [OracleHelpOptions] keyword 391
 [JavaHelp window], assign default parameters 394
 [JavaHelpOptions]
 set-up options and settings 376
 subject to configuration overrides 925
 JavaRootFiles, [JavaHelpOptions]
 keyword 380
 JavaRootFiles, [OracleHelpOptions]
 keyword 380
 [JH2_HelpsetAddition] 383
 [JHImages] 394
 JHVersion2, [JavaHelpOptions] keyword 376

K

KeepCompileWindow, [Automation]
 keyword 334, 388, 390, 420
 activated by CompileHelp 972
 KeepFileNameSpaces, [HTMLOptions]
 keyword 949
 KeepFileNameUnderscores, [HTMLOptions]
 keyword 948

KeepGraphicsInPara, [Graphics] keyword 714
 KeepLink, [HtmlStyles] format property
 fix <\$nopage> index links 444
 make index entries into links 443
 replace page numbers for HTML Help 325
 replace page numbers in index 443
 KeepReplacedCharLinks, [HTMLOptions]
 keyword 443
 KeepXrefText, [DITAOptions] keyword 528
 KeywordLimit, [MSHtmlHelpOptions]
 keyword 212
 KeywordRefs, [Index] keyword 215
 KLink*, [MSHtmlHelpOptions] keywords:
 KLinkButtonGraphic 310
 KLinkButtonHeight 310
 KLinkButtonIcon 310
 KLinkButtonText 310
 KLinkButtonWidth 310
 KLinkEmptyTopic 310
 KLinkFlags 310
 KLinkText 310
 KLinkTextFont 310
 KLinkType 310

L

Last*, [Inserts] keywords:
 LastBottom 600
 position trails of links 631
 LastEnd 600
 LastFrames 600
 LastHead 600
 LastHeadEnd 600
 LastTop 600
 Left
 [HTMLParaStyles] format property 656
 [JavaHelp window] parameter 394
 LeftWide, [OmniHelpOptions] keyword 353
 LEnd, [HTMLParaStyles] format property 675, 676
 LevelBreakForSee, [Index] keyword 215
 LFirst, [HTMLParaStyles] format property
 for lists with multiple paragraph formats 676
 start list style 675
 Link*, [MSHtmlHelpOptions] base keywords:
 LinkButtonGraphic 310
 LinkButtonHeight 310
 LinkButtonIcon 310

ABCDEFGHIJKLMNOPQRSTUVWXYZ

LinkButtonText 310
 LinkButtonWidth 310
 LinkEmptyTopic 310
 LinkFlags 310
 LinkText 310
 LinkTextFont 310
 LinkType 310
 LinkClass, [HTMLParaStyles] WAI keyword 759
 LinkClassIsParaClass, [CSS] keyword 611
 default depends on UseCSS 688
 LinkLogAlways, [Setup] keyword: display broken-link log 112, 428
 LinkSrc
 [HTMLCharStyles] format property 823
 [HTMLParaStyles] format property 612, 823
 [XrefStyles] format property 618
 LinkTitle, [HTMLParaStyles] WAI keyword 758
 ListMissingRefs, [HTMLOptions] keyword 618
 ListN, [HTMLParaStyles] format property 675
 ListType
 [EclipseHelpOptions] keyword 411
 [JavaHelpOptions] keyword 207, 320
 set-up option 375
 [MSHtmlHelpOptions] keyword 207, 320
 set-up option 299
 [OmniHelpOptions] keyword 357
 LLevel, [HTMLParaStyles] format property 675, 677
 LNest, [HTMLParaStyles] format property 675, 677
 LocalConfigPath, [Setup] keyword
 set-up option 60
 [LocalTOC]
 code for local-TOC entries 632, 633
 subject to configuration overrides 925
 where to place local TOCs 631
 LocalTOC*, [LocalTOC] keywords:
 LocalTOCEnd 632
 LocalTOCItem 633
 LocalTOCStart 632
 LocalTOCSubs 633
 [LocalTOCLevels] 632
 subject to configuration overrides 927
 LogAuto, [Automation] keyword 956
 LogDebug, [Logging] keyword 116
 LogErrors, [Logging] keyword 115

LogFileName, [Logging] keyword 115
 [Logging] conversion events 115
 LogInfo, [Logging] keyword 116
 LogIniChains, [Logging] keyword 116
 LogQuerys, [Logging] keyword 116
 LogWarnings, [Logging] keyword 116
 Longdesc, [HTMLParaStyles] format property for
 longdesc attribute 757
 LowerCaseCSS, [CSS] keyword 692
 LowMem, [OmniHelpOptions] keyword 348

M

[MacroFonts], map characters in special fonts 663
 MacroNestMax, [Macros] keyword 792, 816
 Macros, [Templates] keyword 793, 851
 [Macros]
 configure macro definitions for XML 470
 debug 820
 loop-control limits 816
 remove implicit line breaks 789
 subject to configuration overrides 925
 [MacroVariables] 797
 create a macro variable 796
 MacroVarNesting, [Macros] keyword 798
 MadeWith*, [HtmlOptions] keywords:
 MadeWithAttributes 453
 MadeWithImageFile 452
 MadeWithLink 453
 MadeWithPara 453
 MainCssName, [OmniHelpOptions] keyword 350
 set-up option 347
 MakeAliasFile, [MSHtmlHelpOptions]
 keyword 330
 set-up option 299
 MakeALinkFile, [OracleHelpOptions]
 keyword 400
 MakeCshMapFile, [MSHtmlHelpOptions]
 keyword 329
 MakeFileHrefsLower, [HTMLOptions]
 keyword 406, 613
 for JavaHelp 391
 MakeLocalTOC, [LocalTOC] keyword 631
 MakeTrail, [Trails] keyword 628
 enable [HTMLParaStyles]Trail format

ABCDEFGHIJKLMNOPQRSTUVWXYZ

property 628

MapBookRelTable, [DITAOptions] keyword 550

MapBookTopics, [DITAOptions] keyword 540, 550

MapFilePrefix, [JavaHelpOptions]
keyword 381
set-up option 376

MapFilePrefix, [OracleHelpOptions]
keyword 381

MapHead, [DITAOptions] keyword 542

MapID, [DITAOptions] keyword 543

MapName, [DITAOptions] keyword 541

MapTitle, [DITAOptions] keyword 541

MapTopicID, [DITAOptions] keyword 543
default for FrameMaker 8 import 481

MapTopicMeta, [DITAOptions] keyword
default for FrameMaker 8 import 481

MapTopicmeta, [DITAOptions] keyword 543

[Markers], invent and clone marker types 139, 836, 837

[MarkerTypeCodeAfter] 843

[MarkerTypeCodeBefore] 843

[MarkerTypeCodeReplace] 843

[MarkerTypes], marker-type properties 838

MathFullForm, [HTMLOptions] keyword 518

MergeFirst, [OmniHelpOptions] keyword 369

MergePre, [HTMLOptions] keyword 438

Meta
[HTMLOptions]GeneratorTag property 433
[HTMLParaStyles] format property 434, 598

MidHigh, [OmniHelpOptions] keyword 353

ModelName, [Topic] keyword 907, 909

MoveArchive, [Automation] keyword 976

[MSHtmlHelpOptions] 300–340
set-up options and settings 299
subject to configuration overrides 925

MultiImageFigures, [DITAOptions]
keyword 517

MultiImageFigures, [DocBookOptions]
keyword 581

MultipleOLE, [GraphExport] keyword 882

N

N4CssName, [OmniHelpOptions] keyword 350

N4CtrlCssName, [OmniHelpOptions]
keyword 352

N6CssName, [OmniHelpOptions] keyword 350

N6CtrlCssName, [OmniHelpOptions]
keyword 352

Name, [JavaHelp window] parameter 394

NameUndefinedMacros, [Macros] keyword 820

NameUndefinedMacroVars, [Macros]
keyword 820

NavElems, [OmniHelpOptions] keyword 356
set-up option 346

NavIcons, [JavaHelp window] parameter 394
[NavigationMacros]
subject to configuration overrides 925
use buttons for 638
635

NavPane, [JavaHelp window] parameter 394

NestTopicFiles, [DITAOptions] keyword 521

NewWindow, [OmniHelpOptions] keyword 352

NextButton, [NavigationMacros] keyword 640

NextFSButton, [NavigationMacros]
keyword 641

NextFSMacro, [NavigationMacros]
keyword 639, 828

NextMacro, [NavigationMacros] keyword 639, 828

NoAccess, [TableAccess] property 765

NoAnum, [HTMLParaStyles] format property, remove autonumbers 465, 649
from footnotes 673

NoAnum, [HtmlStyles] format property, remove autonumbers
for XML 466

NoAttribLists, [CSS] keyword 678

NoBreak, [HtmlStyles] format property 651

NoClassLists, [CSS] keyword 678
default depends on UseCSS 688

NoColID, [HTMLParaStyles] WAI format
property 768

NoColor, [HTMLParaStyles] or
[HTMLCharStyles] format property 657, 669

ABCDEFGHIJKLMNOPQRSTUVWXYZ

NoCondAttrs, [DITAOptions] keyword [534](#)

NoContLink, [HTMLParaStyles], HTML Help format property [322](#), [323](#)

NoCSS, [HTMLParaStyles] or [HTMLCharStyles] format property [700](#)

NoFig, [HTMLParaStyles] or [HTMLCharStyles] format property, prevent wrapping DITA image in <fig> [517](#)

NoFonts, [HTMLOptions] keyword [665](#)
 CSS-dependent default value [688](#)
 prevent tags from overriding CSS [701](#)
 XML default value [459](#)

NoFootnoteLinks, [HTMLOptions] keyword [672](#)

NoFrameAbove, [HtmlStyles] format property [651](#), [712](#)

NoFrameBelow, [HtmlStyles] format property [651](#), [712](#)

NoHref, [HtmlStyles] property [444](#)

NoLocations, [HTMLOptions] keyword [614](#)

NoMemDel, [Options] keyword [1036](#)

NoNameDel, [Options] keyword [1036](#)

NonsplitBottom, [Inserts] keyword [600](#)

NonsplitTop, [Inserts] keyword [600](#)

NoPara, [HTMLParaStyles] format property, strip <p> tags [650](#)
 for XML [464](#)

NoPara, [HtmlStyles] format property, strip <p> tags
 render text frame as in-line text [649](#)

NoParaClose, [HTMLOptions] keyword [437](#)

NoRef
 [HTMLParaStyles] format property [414](#), [614](#)
 [XrefStyles] format property [618](#)

NoSize, [HTMLParaStyles] or [HTMLCharStyles] format property [657](#)

NoSplit
 [HTMLParaStyles] format property [590](#)

NoSymbolFont, [HTMLOptions] keyword [666](#)

NoTags, [HTMLParaStyles] or [HTMLCharStyles] format property [650](#)

NoTags, [HtmlStyles] format property
 render text frame as in-line text [649](#)

NoWrap, [HTMLOptions] keyword [438](#), [461](#)

NoWrap, [HTMLParaStyles] format property, suppress line breaks [650](#)

NumericCharRefs, [HTMLOptions] keyword [432](#)
 for Eclipse Help [412](#)
 for XML [460](#)

O

ObjectIDs, [HTMLOptions] keyword [620](#)
 convert lists of paragraph references [216](#), [325](#), [445](#)

[OHMergeFiles], map OmniHelp projects [367](#)

OHProjFileSuffix, [OmniHelpOptions] keyword [348](#)

OHProjFileXhtml, [OmniHelpOptions] keyword [342](#)

[OHTopLeftNav], code for OmniHelp [353](#)

[OHTopRightNav], code for OmniHelp [353](#)

OHVFiles, [OmniHelpOptions] keyword [370](#)

OHViewPath, [OmniHelpOptions] keyword [343](#), [370](#)

OmitMacroReturns, [Macros] keyword
 ignore line breaks in macros, for XML [470](#)
 omit line breaks in macro output [789](#)

[OmniHelpOptions] [347–369](#)
 subject to configuration overrides [925](#)

OnlyAuto, [Automation] keyword [977](#)

OpenlinkIsFile, [HtmlOptions] keyword [620](#)

[Options]
 for cases, spaces, and wildcards [113](#)
 for conversion debugging [1036](#)
 subject to configuration overrides [925](#)

[OracleHelpWindows] [398](#)

Overrides, [HtmlStyles] format property
 override paragraph properties [657](#)

P

Padding, [Table] keyword [740](#)
 overridden by [TableAttributes] [736](#), [740](#)
 set-up option [426](#)

PageBreaks, [HTMLOptions] keyword, for split points [587](#)

[ParaClasses]
 default use for CSS class names [682](#)
 map paragraph formats [693](#)
 map paragraph formats for XML [463](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- subject to configuration overrides 925
- ParaLink, [HtmlStyles] format property 138
 - make index entries into links 443
 - make LOM entries into links 325
 - make LOP entries into links 325
- ParaLinkClass, [HTMLParaStyles] format property 611
- [ParaStyle*] sections
 - [ParaStyleCode*] sections
 - all subject to configuration overrides* 927
 - [ParaStyleCodeAfter] 467, 823
 - [ParaStyleCodeBefore] 614, 823
 - [ParaStyleCodeEnd] 823
 - [ParaStyleCodeReplace] 823
 - [ParaStyleCodeStart] 799, 823
 - [ParaStyleCSS] 700
 - [ParaStyleLinkSrc] 612, 823
- [ParaTags]
 - assign HTML tags to paragraph formats 647
 - assign XML tags to paragraph formats 463
 - designate script paragraph formats 650
 - map format names to CSS class names 693
 - subject to configuration overrides 927
 - tags used for CSS classes by default 682
- PersistSettings, [OmniHelpOptions] keyword 354
- PixelSpacerImage, [HTMLOptions] keyword 716
 - indent images 716
 - indent tables 747
- Plain, [HTMLParaStyles] or [HTMLCharStyles] format property 657
- PluginID, [EclipseHelpOptions] keyword 408
- PluginID, [EclipseHelpOptions] keyword
 - set-up option 405
- PluginName, [EclipseHelpOptions]
 - keyword 408
 - set-up option 405
- PluginProvider, [EclipseHelpOptions] keyword 408
- PluginSchemaVersion, [EclipseHelpOptions] keyword 410
- PluginVer, [EclipseHelpOptions] keyword 408
- PluginVersion, [Setup] keyword 1034
 - determined at run time 863
- Pop*
 - [JavaHelpOptions] keywords:
 - PopFontColor 397
 - PopFontFamily 396
 - PopFontSize 397
 - PopFontStyle 397
 - PopFontWeight 397
 - PopGraphic 397
 - PopMarkerPrefix 398
 - PopSize 396
 - PopText 396
 - PopType 396
 - [MSHtmlHelpOptions] keywords:
 - PopColors 306
 - PopFont 306
 - PopMargins 306
- pre, [ParaTags] format property 647
- PrevButton, [NavigationMacros] keyword 640
- PrevFSButton, [NavigationMacros] keyword 641
- PrevFSMacro, [NavigationMacros] keyword 639, 828
- PrevMacro, [NavigationMacros] keyword 639, 827
- PrjFileName, [Setup] keyword 1026
 - determined at run time 863
- ProjectName, [OmniHelpOptions] keyword 347
- ProjectTemplate, [OmniHelpOptions] keyword 356
- PrologDTD, [Topic] content-model keyword 909
- PrologDType, [Topic] content-model keyword 909

Q

QuotedEncoding, [HTMLOptions] keyword 432

R

- Raw, [HTMLParaStyles] format property
 - for ALink paragraphs 313
 - for marker-only paragraphs 842
 - for split files 595
- Raw, [HTMLParaStyles] or [HTMLCharStyles] format property 650
- ReAnchorFrames, [HTMLOptions] keyword 713, 886
- [RefFiles], link paths 623

ABCDEFGHIJKLMNOPQRSTUVWXYZ

RefFileType
 [EclipseHelpOptions] keyword 412
 [JavaHelpOptions] keyword 207
 [MSHtmlHelpOptions] keyword 207, 320
 [OmniHelpOptions] keyword 207, 357
 RefPageGraphIndent, [Graphics] keyword 713
 RemoveAHrefAttrs, [HTMLOptions] XML
 keyword 467
 RemoveANames, [HTMLOptions] keyword 614
 for XML link anchors 467
 RemoveATags, [HTMLOptions] XML keyword 467
 RemoveChmFilePaths, [MSHtmlHelpOptions]
 keyword 338
 RemoveEmptyParagraphs, [HTMLOptions]
 keyword 652
 RemoveEmptyTableParagraphs, [Table] keyword
 for DITA 513
 for DocBook 564
 for HTML 744
 RemoveFilePaths, [HTMLOptions] keyword 618
 identify links to other files 349, 622
 RemoveFramesAbove, [HTMLOptions]
 keyword 651
 RemoveFramesBelow, [HTMLOptions]
 keyword 651
 RemoveInternalAnchors, [JavaHelpOptions]
 keyword 385, 399
 RemoveInternalAnchors, [OracleHelpOptions]
 keyword 385, 399
 RemoveWordTocMarkers, [HTMLOptions]
 keyword 114
 RemoveXrefHotspots, [HTMLOptions]
 keyword 468, 563
 RepeatMax, [Macros] keyword 817
 ReplaceFrameVars, [Macros] keyword 123
 RetainRuninImagesForEmptyParagraphs,
 [Graphics] keyword 714, 744
 Right, [HTMLParaStyles] format property 656
 RowAttribute, [HTMLParaStyles] format
 property 738
 RowGroup, [HTMLParaStyles] WAI format
 property 768, 769
 in header cells to define row groups 767
 in [Table]RowGroupHead cells 778
 use with scope method 780

RowGroupHead, [Table] WAI keyword 778
 id/header table cell attribute 778
 row-group extent 779
 RowGroupIDs, [Table] WAI keyword 779
 id/header table cell attribute 778
 override for selected tables 784
 row-group extent 779
 with id/headers method 767
 RowHead, [Table] WAI keyword 782
 id/header table cell attribute 778
 row extent 783
 RowIDs, [Table] WAI keyword 782
 id/header table cell attribute 778
 override for selected tables 784
 row extent 783
 RowSpanHead, [Table] WAI keyword 781
 id/header table cell attribute 778
 row-span extent 781
 RowSpanIDs, [Table] WAI keyword 780
 dependencies 765
 id/header table cell attribute 778
 override for selected tables 784
 row-span extent 781
 RunfmDiagnostics, [Automation] keyword 957,
 988
 RunInHeads, [HTMLOptions] keyword 648
 for Help systems 203

S

Scope
 [HTMLParaStyles] WAI format property 768,
 769
 [Table]AccessMethod property 764
 apply scope method to all tables 764
 avoid redundant attributes 456
 dependencies 776, 785
 enable [Table*Attributes] 751
 [TableAccess] property 765
 enable [Table*Attributes] 751
 Scope, [Templates] keyword 855, 861
 ScopeCol, [Table] WAI keyword 776
 dependencies 765
 override for selected tables 784
 ScopeColGroup, [Table] WAI keyword 776
 dependencies 765
 override for selected tables 784

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- ScopeRow, [Table] WAI keyword 776
 - dependencies 765
 - override for selected tables 784
- ScopeRowGroup
 - [Table] WAI keyword
 - dependencies 765
 - enable [Table*Attributes] 751
 - override for selected tables 784
 - [TableAccess] override, enable
 - [Table*Attributes] 751
 - [Tables] WAI keyword 776
- script, [ParaTags] format property 647
- ScriptType, [HTMLOptions] keyword 650
- SearchHighlightStyle, [OmniHelpOptions]
 - keyword 364
- SearchWordMin, [OmniHelpOptions]
 - keyword 363
- Sec*, [JavaHelpOptions] secondary-window
 - properties
 - SecFontColor 397
 - SecFontFamily 396
 - SecFontSize 397
 - SecFontStyle 397
 - SecFontWeight 397
 - SecGraphic 397
 - SecLocation 396
 - SecName 396
 - SecSize 396
 - SecText 396
 - SecType 396
- SecMarkerPrefix, [JavaHelpOptions]
 - keyword 398
- [SecWindows], secondary windows 225
 - for HTML Help 318
 - for OmniHelp 360
 - for Oracle Help 399
 - subject to configuration overrides 927
- SeeAlsoTerm, [Index] keyword 214
- SeeTerm, [Index] keyword 214
- SelectorIncludesTag, [CSS] keyword 696
- SetElementIDs, [DITAOptions] keyword 496
- SetFrameConditions, [Setup] keyword 123
- [Setup]
 - conversion-template settings 81, 864, 865
 - convert generated files for ASCII DCL 1011
 - convert system variables to text 437
 - convert TOC and IX 124
 - for ASCII DCL 1011
 - for Help systems 205
- equations 137, 726, 884
- exclude generated files 125
- export options and settings 84
- file names 120, 623
- FrameMaker-exported graphics 130, 133, 134, 725, 726, 884, 885
- generate/update 126
- manage MIF files 111
- options determined at run time, *listed* 863
- set-up options and settings 81
- subject to configuration overrides 925
- ShipPath, [Automation] keyword 975
 - activated by CompileHelp or FTSCCommand 972
 - activated by WrapAndShip 956
- ShowLog, [Logging] keyword 115
- ShowNavLeft, [OmniHelpOptions] keyword 354
- ShowSubjects, [OmniHelpOptions] keyword 360
- ShowUndefinedFormats, [Logging] keyword 116
- Size, [Base] keyword 664
- SizeN, [HTMLParaStyles] format property 656
 - overrides [FontSizes] 665
- SmartSplit, [HTMLOptions] keyword 590
 - do not use with local TOCs 631
- SortSeeAlsoFirst, [Index] keyword 215
- SpacelessMatch, [Options] keyword 104, 113
- [Spacer], indent images and tables 793
- SpacerAlt, [HTMLOptions] keyword 716
- Spacing, [Table] keyword 740
 - overridden by [TableAttributes] 736, 740
 - set-up option 426
- Span, [HTMLParaStyles] WAI format
 - property 768, 769
 - identify rows and columns 781
- SpecIniDir, [DITAOptions] keyword 916
- Split
 - [HTMLParaStyles] format property 586
 - trail dependencies 628
 - [HtmlStyles] format property
 - for local-TOC level numbers 632
 - retains ObjectIDs for TOC 445
 - [MarkerTypes] property 587, 839
- Split*, [Inserts] split-file keywords:
 - SplitBottom 600
 - position local TOCs 634

ABCDEFGHIJKLMNOPQRSTUVWXYZ

position trails of links 631
 SplitEnd 600
 SplitFrames, 600
 SplitHead 600
 SplitHeadEnd 600
 SplitTop 600
 SplitTopicFiles, [DITAOptions] keyword 520
 SplitTrail, [Trails] keyword 630
 dependencies 628
 StartingFSButton, [NavigationMacros]
 keyword 641
 StartingPrevFSButton, [NavigationMacros]
 keyword 641
 StartingPrevFSMacro, [NavigationMacros]
 keyword 639, 828
 StartingSplit, [HTMLOptions] keyword 588
 prevent splits that leave dangling headings 590
 [StopWords], for OmniHelp search 363
 Strike, [HTMLCharStyles] or
 [HTMLCharStyles] format property 657
 StripGraphPath, [Graphics] keyword 704, 887
 locate replacements for EPS graphics 877
 synchronize graphics settings 968
 use referenced graphics without converting 706
 use system commands to manage files 335
 StripTable, [Table] keyword 753
 [Style*] sections
 all subject to configuration overrides 927
 [StyleCellAbbr] 769, 770
 [StyleCellAttribute] 769, 738
 [StyleCellAxis] 769
 [StyleCellScope] 769
 [StyleCode*] sections
 [StyleCodeAfter] 467
 [StyleCodeStore] 804
 [StyleFilePrefix] 951
 [StyleFileSuffix] 951
 [StyleMetaName] 434
 [StyleParaLinkClass] 611
 [StyleRowAttribute] 738
 [StyleTextStore] 803
 [StyleTitlePrefix] 596
 [StyleTitleSuffix] 596
 [StyleTrailPrefix] 628
 [StyleTrailSuffix] 629
 [StyleWindow] 318
 [StyleTabReplace], replace tabs with code 659

Summary, [HTMLParaStyles] WAI table
 keyword 761
 SystemCommandWindow, [Automation]
 keyword 939
 SystemEndCommand, [Automation] keyword 938
 SystemStartCommand, [Automation]
 keyword 938
 SystemWrapCommand, [Automation] keyword 938

T

TabCharsPerInch, [HTMLOptions] keyword 671
 [TableAccess], override properties 733, 765, 784
 override [Table] default access method 765
 subject to configuration overrides 928
 [TableAfterMacros] 749
 subject to configuration overrides 928
 TableAttributes, [Table] keyword 464, 739,
 741
 XML default value 459
 [TableAttributes]
 overrides [Attributes] values 736
 overrides border, cellpadding, and
 cellspacing in [Attributes] 736
 overrides [Table]Border, Padding, and
 Spacing 736
 subject to configuration overrides 928
 summary and title 761
 [TableBeforeMacros] 749
 add space before tables 749
 invoke macros around tables 749
 subject to configuration overrides 928
 TableBody, [HTMLParaStyles] WAI table-cell
 property 770
 [TableBodyAttributes] 750
 subject to configuration overrides 928
 TableCaptionTag, [Table] keyword 564, 747
 [TableCellAttributes] 751
 base CSS class on table format 738, 739
 subject to configuration overrides 929
 [TableCellEndMacros] 751
 subject to configuration overrides 929
 [TableCellStartMacros] 751
 selectively modify table text 752
 subject to configuration overrides 929
 [TableClasses], map table formats to CSS

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- classes 694, 737
- TableColsRelative, [DITAOptions]
 - keyword 513
- TableContinued, [Table] keyword 748
- TableContVar, [Table] keyword 748
- TableDPI, [Table] keyword 742
 - control width of table columns 743
 - set-up option 426
- [TableEndMacros]
 - capture row and column counts 751
 - invoke macros around tables 749
 - subject to configuration overrides 929
- [TableFooterAttributes] 750
 - subject to configuration overrides 929
- TableFooterClass, [DITAOptions] keyword 511
- TableFooterRows, [Table] keyword 734
 - overridden by [TableAccess] method 735
- TableFootnoteSeparator, [Table] keyword 748
- TableFootnotesWithTable, [Table]
 - keyword 748
- [TableGroup] 729
 - assign with ***Config** marker 930
 - subject to configuration overrides 929
- TableHead, [HTMLParaStyles] WAI table-cell
 - property 770
- [TableHeaderAttributes] 750
 - subject to configuration overrides 929
- TableHeaderCols, [Table] keyword 734
 - effect on ScopeRow 776
 - overridden by [TableAccess] method 735
- TableHeaderRows, [Table] keyword 734
 - effect on ScopeCol 776
 - overridden by [TableAccess] method 735
- TableIndents, [Table] keyword 747
- [TableIndents] 747
 - subject to configuration overrides 929
 - unindent tables 303
- TableParents, [DITAOptions] keyword 512
- TableParents, [DocBookOptions] keyword 580
- [TableReplaceMacros] 749
 - subject to configuration overrides 929
- [TableRowAttributes] 751
 - subject to configuration overrides 929
- [TableRowEndMacros] 751
 - subject to configuration overrides 929
- [TableRowStartMacros] 751
 - selectively modify table text 752
 - subject to configuration overrides 929
- [Tables]
 - access method 763–773, 776, 780, 785
 - caption 563, 747
 - cell access method 775–784
 - eliminate attributes for XML 464, 739, 741
 - overridden by [TableAttributes] 737
 - properties 740
 - split-file titles 595
 - structure 730–735
 - subject to configuration overrides 925
- TableSheet, [Table] keyword 748
- TableSheetVar, [Table] keyword 748
- TableSizing, [Table] keyword 742
 - control column width 743
 - overridden by [TableSizing] 742
 - set-up option 426
- [TableSizing] 742
 - subject to configuration overrides 929
- [TableStartMacros] 749
 - capture row and column counts 751
 - override column or row groups 733
 - selectively modify table text 752
 - specify <col> elements 732
 - subject to configuration overrides 929
- TableTitle, [HTMLParaStyles] format
 - property 761
- TableTitles, [Table] keyword 747
- [TableUseRowColor] 746
 - subject to configuration overrides 929
- TableWordBreak, [Table] keyword 744
- [TargetFiles] 451
 - for jumps from image maps 725
 - for jumps to a window type 617
- [Targets] 451
 - for jumps to a window type 617
 - subject to configuration overrides 928
- TbFootClass, [CSS] keyword 694
- TemplateFileName, [Setup] keyword 864
 - for chapter-specific templates 865
 - set-up option 81
- [Templates] 859
 - for document-specific settings 854
 - for general configuration settings 851
 - for macro libraries 851

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- Text, [MarkerTypes] property [839](#)
- Text, [XrefStyles] format property [618](#)
- [TextFlows], use or omit [113](#)
- TextFrameIsText, [HtmlStyles] format property [649](#)
- TextInsetMark, [DITAOptions] keyword [534](#)
- TextInsetNest, [DITAOptions] keyword [535](#)
- TextStore, [HTMLParaStyles] format property [803](#)
 - create a macro variable [796](#)
- Title
 - [HTMLOptions] keyword [433](#), [597](#)
 - [HTMLParaStyles] format property [434](#)
 - for split files [595](#)
 - trail dependencies [628](#)
 - [HtmlStyles] format property
 - retains ObjectIDs for TOC [445](#)
 - [JavaHelp window] parameter [394](#)
 - [MarkerTypes] property [840](#)
- [Titles]
 - for individual output files [434](#)
 - overrides [HTMLParaStyles]Title [596](#)
 - precedence [595](#)
- Toc*
 - [EclipseHelpOptions] keywords:
 - TocExtradir [410](#)
 - TocFilename [410](#)
 - TocLabel [413](#)
 - TocLinkTo [416](#)
 - TocNamesFileOnly [414](#)
 - TocPrimary, [410](#)
 - TocTopic [413](#)
 - [JavaHelpOptions] keywords:
 - TocClosedImage [386](#)
 - TocOpenImage [386](#)
 - TocTopicImage [386](#)
 - [OmniHelpOptions] keywords:
 - TocButtons [359](#)
 - TocExpand [358](#)
 - TocGroupsOpen [358](#)
 - TocIcoBase [358](#)
 - TocOpenLevel [358](#)
- TocIdxFilePrefix, [EclipseHelpOptions] keyword [412](#)
- [TocLevelExpand], JavaHelp 2 settings [385](#)
- [TocLevelImage], JavaHelp 2 settings [386](#)
- TocTopic, [EclipseHelpOptions] keyword
 - set-up option [405](#)
- Toolbar, [JavaHelp window] parameter [394](#)
- Top
 - [Inserts] keyword [599](#), [600](#), [821](#)
 - to position a navigation macro [642](#)
 - to position trails of links [631](#)
 - [JavaHelp window] parameter [394](#)
- TopButton, [NavigationMacros] keyword [641](#)
- TopFirst, [OmniHelpOptions] keyword [353](#)
- TopHigh, [OmniHelpOptions] keyword [353](#)
- [Topic], content-model section
 - debug DITA topic types [918](#)
 - prolog for DITA topic type [909](#)
 - specialize DITA topic type [914](#)
- TopicBody, [Topic] content-model keyword [910](#), [913](#)
- TopicBreak, [Inserts] keyword [586](#), [599](#), [821](#)
- TopicDerivation, [Topic] content-model keyword [910](#), [914](#)
- [TopicFirst], content-model section; first-child elements for a DITA topic type [911](#)
- TopicID, [DITAOptions] keyword [526](#)
- [TopicLevels], content-model section; required levels for specialized DITA elements [911](#)
- [TopicParents], content-model section; possible parents of specialized elements [910](#)
- TopicRoot, [Topic] content-model keyword [909](#)
- TopicStart, [Topic] content-model keyword [909](#), [913](#)
- TopicStartCode
 - [MarkerTypes] property [840](#)
- [TopicTables], content-model section; table configuration for DITA topic types [916](#)
- TopMacro, [NavigationMacros] keyword [640](#)
- Trail, [HTMLParaStyles] format property [628](#)
 - dependencies [628](#)
 - for non-heading formats [630](#)
 - required for first paragraph in file [631](#)
- Trail*, [Trails] keywords:
 - TrailCurrent [629](#)
 - TrailEnd [629](#)
 - TrailIndent [629](#)
 - TrailPosition [630](#)
 - TrailSep [629](#)
 - TrailStart [629](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

[TrailLevels] 630
 subject to configuration overrides 928
 [Trails], bread-crumbs link list 628–630
 subject to configuration overrides 925
 [Typographics] 493, 667

U

Uline, [HTMLCharStyles] or [HTMLCharStyles]
 format property 657
 UnicodeFTS
 [OmniHelpOptions] keyword 362
 UnicodeLocale
 [OmniHelpOptions] keyword 362
 UnwrapPRE, [HTMLOptions] keyword 437, 670
 URLTarget, [HTMLOptions] keyword 625
 UseAliasAName, [MShtmlHelpOptions]
 keyword 327
 UseAltMapTitle, [DITAOptions] keyword 542
 UseAltShading, [Table] keyword 745
 UseAnums, [HTMLOptions] keyword
 for HTML output 649
 for XML output 465
 XML default value 459
 UseBackForward, [OmniHelpOptions]
 keyword 354
 UseCALSMModel, [Table] keyword 563, 730
 XML default value 459
 UseCharacterTypographics, [Typographics]
 keyword 667
 UseCharRangeClasses, [CSS] keyword 695
 UseChmInLinks, [MShtmlHelpOptions]
 keyword 308
 UseCodePage
 [MShtmlHelpOptions] keyword 300
 UseCommaAsSeparator, [Index] keyword 213
 UseCommonNames, [DITAOptions] keyword 504
 UseCompositeDropJS, [DropDowns] keyword 231
 UseContext, [EclipseHelpOptions]
 keyword 411
 set-up option 405
 UseCSS, [CSS] keyword 684
 affects default value of
 [Graphics]GraphAlignAttributes 715
 [HTMLOptions]AlignAttributes 657
 [HTMLOptions]Basefont 664
 LinkClassIsParaClass 611
 NoClassLists 678
 affects default values of other settings 688
 affects use of tags 688, 701
 replaces [HtmlOptions]Stylesheet 687
 set-up option 426, 479, 561
 UseCSSLeading, [HTMLOptions] keyword 701
 UseDefaultStopWords, [OmniHelpOptions]
 keyword 363
 UseDOCTYPE, [HTMLOptions] keyword 429, 563
 UseDoneDialog, [Setup] keyword 113
 UseDropDowns, [DropDowns] keyword 227
 UseDTDPath, [DITAOptions] keyword 481
 UseExistingDCL, [Setup] keyword 111
 determined at run time 863
 export embedded graphics 131
 export option 84
 UseExistingMIF, [Setup] keyword 111
 determined at run time 863
 export option 84
 UseFavorites, [JavaHelpOptions] keyword 383
 UseFileIDs, [HTMLOptions] keyword 120
 identify links to other files 621
 UseFontFace, [HTMLOptions] keyword 664, 666
 UseFontSize, [HTMLOptions] keyword 666
 UseFootnoteLists, [HTMLOptions] keyword 673
 UseFootXrefTag, [HTMLOptions] keyword 564,
 672
 XML default value 459
 UseFormatAsTag
 [DocBookOptions] keyword 565
 DITAOptions] keyword 487
 UseFormatTypographics, [Typographics]
 keyword 667
 UseFrameGenFiles, [Setup] keyword
 for ASCII DCL output 1011
 include generated IOM files 442
 include generated list files 444
 omit generated files 125
 set-up option 81
 UseFrameIX, [Setup] keyword 124
 for ASCII DCL output 1011
 for Help systems 205
 set-up option 81
 UseFrameSet, [HTMLOptions] keyword 451

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- UseFrameTOC, [Setup] keyword [124](#)
 - for Help systems [205](#)
 - include generated files [1011](#)
 - set-up option [81](#)
- UseFTS
 - [EclipseHelpOptions] keyword [411](#)
 - [JavaHelpOptions] keyword [387](#)
 - set-up option [375](#)
 - [MSHtmlHelpOptions] keyword [326](#)
 - set-up option [299](#)
 - [OmniHelpOptions] keyword [362](#)
 - set-up option [347](#)
- UseGlossary, [JavaHelpOptions] keyword [392](#)
- UseGraphicFileID, [Setup] keyword [133](#)
 - name converted graphics [885](#)
 - name files from FrameMaker export filters [726](#)
- UseGraphicPreviews, [Graphics] keyword [130](#), [884](#)
 - determined at run time [863](#)
 - export option [84](#)
 - turn off for replaced graphics [131](#)
- UseHash, [HTMLOptions] keyword [467](#), [563](#)
- UseHeadAndBody, [HTMLOptions] keyword [435](#), [461](#)
 - XML default value [459](#)
- UseHideShow, [OmniHelpOptions] keyword [354](#)
- UseHVIndex, [Index] keyword [211](#)
- UseHyperColor, [HTMLOptions] keyword [139](#)
- UseIndex, [EclipseHelpOptions] keyword [410](#)
 - set-up option [405](#)
- UseIndexentryTag, [JavaHelpOptions] keyword [386](#)
- UseIndexentryTag, [OracleHelpOptions] keyword [386](#)
- UseInformaltableTag, [Table] keyword [563](#), [747](#)
- UseInitDialog, [Setup] keyword [113](#)
- UseListButton, [OmniHelpOptions] keyword [354](#)
- UseListedXrefFilesOnly, [HTMLOptions] keyword [468](#), [563](#)
- UseListTypeAttribute, [CSS] keyword [384](#), [678](#)
- UseLocalFileID, [Setup] keyword [120](#), [122](#)
- UseLocalScope, [DITAOptions] keyword [529](#)
- UseLog, [Logging] keyword [115](#)
- UseManifest, [EclipseHelpOptions] keyword [407](#)
- UseMapID
 - DITAOptions] keyword [542](#)
- UseNavButtons, [NavigationMacros] keyword [638](#)
- UseOriginalGraphicNames, [Graphics] keyword [889](#)
 - determined at run time [863](#)
 - export option [84](#)
 - for referenced graphics [884](#)
 - for replaced graphics files [131](#)
 - for unconverted graphics files [706](#), [889](#)
- UseOutputClass, [DITAOptions] keyword [498](#)
- UseParagraphTypographics, [Typographics] keyword [667](#)
- UsePlugin, [EclipseHelpOptions] keyword [406](#)
- UsePrevNext, [OmniHelpOptions] keyword [354](#)
- UsePtSuffix, [Graphics] keyword
 - default for FrameMaker 8 import [481](#)
- UsePxSuffix, [Graphics] keyword [722](#)
- UseRawName, [HTMLOptions] keyword [948](#)
- UseRawNewlinks, [HTMLOptions] keyword [241](#), [401](#)
- UseRelNameColumn, [DITAOptions] keyword [546](#)
- UseRowColor, [Table] keyword [745](#)
- UseRunInTag, [HTMLOptions] keyword [648](#)
- UseRuntime
 - [EclipseHelpOptions] keyword [411](#)
 - [UserVarPrompts], user variables [942](#)
 - [UserVars], user variables [941](#)
 - create a macro variable [796](#)
- UseSearchHighlight, [OmniHelpOptions] keyword [364](#)
- UseSingleton, [EclipseHelpOptions] keyword [408](#)
- UseSortString, [Index] keyword [218](#)
- UseSpacers, [HTMLOptions] keyword [716](#), [747](#)
- UseSpanAsDefault, [CSS] keyword [693](#)
- UseStart, [OmniHelpOptions] keyword [354](#)
- UseSubHelpSets, [JavaHelpOptions] keyword [400](#)
- UseSubHelpSets, [OracleHelpOptions] keyword [400](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

UseTableFooterClass, [DITAOptions]
keyword 511

UseTbFootnoteLists, [HTMLOptions]
keyword 673

UseTbHeaderCode, [Table] keyword 731

UseTitleForAlt, [Graphics] keyword 724

UseTopButtons, [OmniHelpOptions]
keyword 353

UseTopicAlias
DITAOptions] keyword 485

UseTopicAlias, [DITAOptions] keyword 536

UseTypographicElements, [Typographics]
keyword 493, 667

UseTypographicStyles, [Typographics]
keyword 667

UseUlink, [HTMLOptions] keyword 468, 563

UseXMLbr, [HtmlOptions] keyword 465

UseXMLDeclaration, [HTMLOptions]
keyword 436

UseXMLRoot, [HTMLOptions] keyword 435, 563

V

ValidOnly, [HTMLOptions] keyword 454, 658
for Eclipse Help 412

VariableElement, [DITAOptions] keyword 531

VariableFile, [DITAOptions] keyword 531

VariableTopicID, [DITAOptions] keyword 531

VariableType, [DITAOptions] keyword 530

W

WhileMax, [Macros] keyword 816

Width, [JavaHelp window] parameter 394

WildcardMatch, [Options] keyword 113

Window
[MarkerTypes] property 840

Window, [HTMLParaStyles] format property 318

Windows, [JavaHelpOptions] keyword
list of windows 394

WrapAndShip, [Automation] keyword 956
determined at run time 863
export option 84

WrapCopyFiles, [Automation] keyword 962

activated by CompileHelp or FTSCCommand 972

WrapPath, [Automation] keyword 961
activated by CompileHelp or FTSCCommand 972
activated by WrapAndShip 956
for JavaHelp, Oracle Help 379, 382

WrapTopicFiles, [DITAOptions] keyword 521
default for FrameMaker 8 import 481

WriteAllGraphics, [Setup] keyword 130, 131, 884
determined at run time 863
export option 84
third-party graphics tools 131

WriteBookFile, [DocBookOptions] keyword 562

WriteClassAttributes, [CSS] keyword 684
default depends on UseCSS 688
replaces [HtmlOptions] Stylesheet 687
turn off for XML 463, 696

WriteContext, [EclipseHelpOptions]
keyword 411
set-up option 405

WriteCssLink, [CSS] keyword 684
change CSS mid-document 689
customize CSS link tag 690
default depends on UseCSS 688
replaces [HtmlOptions] Stylesheet 687
select CSS file at run time 689
use with CssFileName 686

WriteCssStylesheet, [CSS] keyword 684
default depends on UseCSS 688
designate CSS file 686
generate CSS file 684
replaces [HtmlOptions] Stylesheet 687
set-up option 426, 479, 561

WriteDitamaps, [DITAOptions] keyword 540

WriteDropIconFiles, [DropDowns] keyword 231

WriteDropJSFile, [DropDowns] keyword 235

WriteEquations, [Setup] keyword 136
determined at run time 863
export embedded graphics 131
export option 84
turn off for FrameMaker export 884

WriteHelpProjectFile, [MSHtmlHelpOptions]
keyword 301
set-up option 299

WriteHelpSetFile, [JavaHelpOptions]
keyword 383
set-up option 376

ABCDEFGHIJKLMNOPQRSTUVWXYZ

WriteHelpSetFile, [OracleHelpOptions]
keyword [383](#)

WriteMadeWithGraphic, [HtmlOptions]
keyword [452](#)

WriteManifest, [EclipseHelpOptions]
keyword [408](#)

WriteMasterPageGraphics, [Setup]
keyword [885](#)

WritePlugin, [EclipseHelpOptions]
keyword [409](#)
set-up option [405](#)

WriteRefPageGraphics, [Setup] keyword [885](#)

WriteSpacerFile, [HTMLOptions] keyword [716](#)

WriteVariableFile, [DITAOptions]
keyword [531](#)

X

XHLangAttr, [HTMLOptions] keyword [430, 563](#)

XHLanguage, [HTMLOptions] keyword [430](#)

XHNamespace, [HTMLOptions] keyword [430](#)

XMLBreak, [HtmlStyles] format property [465](#)

XMLBreakPara, [HtmlOptions] keyword [465](#)

XMLEncoding, [HTMLOptions] keyword [460](#)
for double-byte characters [432](#)

XMLGraphAttrs, [HTMLOptions] keyword [468](#)

XMLLinkAttrs, [HTMLOptions] keyword [467](#)

XMLNoBreak, [HtmlStyles] format property [466](#)

XMLRoot, [HTMLOptions] keyword [435, 461](#)
XML default value [459](#)

XMLVersion, [HTMLOptions] keyword [460](#)

[XrefFiles], interfile links [622](#)
map links to text insets [624](#)

XrefFormatIsXrefClass, [CSS] keyword [696](#)
default depends on UseCSS [688](#)
for DITA XML [485](#)

XrefSpaceChar, [HTMLOptions] keyword [613](#)

[XrefStyleLinkSrc], macro for href
attribute [619](#)
for KeyHelp pop-ups [306](#)
subject to configuration overrides [928](#)

[XrefStyles], cross-reference format [618](#)
for KeyHelp pop-ups [306](#)
subject to configuration overrides [928](#)

XrefType, [HTMLOptions] keyword [618](#)

XrefWrapClass, [DITAOptions] keyword [528](#)

Y

No entries for this letter

Z

ZeroCSSMargins, [CSS] keyword [463](#)

ZipCommand, [EclipseHelpOptions]
keyword [419](#)

ZipParams, [EclipseHelpOptions] keyword [419](#)

Subject index

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A

- `` tags, suppressing line breaks in 437
- abbr, HTML table attribute for WAI 1013
 - guidelines for using 759
 - via `CellAbbr` marker 772
 - via `[HtmlStyleCellAbbr]` 769
 - via paragraph format 768
 - via special paragraph format 772
- absolute vs. relative paths
 - in configuration settings 105
 - in graphics references 705
- accented characters, converting to HTML 660
- active portion of a link, defining 139
 - for WinHelp 226
- ActiveX and MS HTML Help 201
- adaptive table sizing
 - for HTML 742
 - overriding 742
 - for WinHelp 261
- `<address>`, HTML paragraph tag 647
- Adobe PDF printer, configuring for `runfm` 985
- after, macro string operator 818
- alert** markers
 - for HTML Help pop-ups 226, 306
 - for HTML split points 587
 - for WinHelp pop-ups 277
 - how to insert 935
- alert** pop-ups, creating for WinHelp 277
- alerttitle** markers
 - for WinHelp pop-ups 277
 - how to insert 935
- alias files for context-sensitive Help 240
- align, HTML attribute
 - and `valign`, automatically generated, excluding from table cells 464, 739
 - eliminating from paragraph tags 656
 - for HTML Help contents entries 322
- aligning
 - graphics for HTML 714
 - headers/footers with graphics for RTF 155
 - sideheads to body paragraphs for print RTF 160
- aligning for HTML 715
- ALink
 - See also* [ALinks](#)
 - jumps, configuring for HTML-based Help 223
 - keywords
 - adding with format properties 222
 - adding with markers 221
 - list destinations, specifying 224
- ALink**, custom marker for Help systems 832
- ALinks
 - See also* [ALink](#)
 - DITA, in relationship tables 546
 - HTML Help
 - creating 309
 - target-and-jump 312
 - uncompiled 310
 - OmniHelp, support for 359
 - Oracle Help for Java, creating 399
 - target-and-jump, for HTML-based Help 224
 - understanding 220
 - WinHelp
 - adding footnotes to topics 285
 - configuring 285
- alt, HTML attribute
 - empty, omitting 718
 - for drop-down icons 231
 - for image maps 723
 - for images
 - purpose and valid content 1015
 - via FrameMaker object attribute 896
 - via **GraphAlt** marker 757
 - via graphic file name 718
 - via special format 757
 - WAI guidelines for using 757
 - for links, via special format 758
- Altura
 - graphics format for WinHelp 264
 - QuickHelp, specifying for WinHelp 247
- anchor paragraph, using to add space, HTML 717
- anchor tags, suppressing line breaks in 437
- anchored frames
 - hiding borders of 718, 723
 - object attributes of, specifying 896
 - positioning
 - for HTML 714
 - for print RTF 191

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- for WinHelp 264
 - tables in, converting to RTF 128
- anchors, internal, for JavaHelp and Oracle Help 399
- ANSI**, custom marker for Help systems 832
- Anum, [HtmlStyles] format property, retain autonumbers 465
- archiving files for delivery 971
- arithmetic operators for macro expressions, *listed* 812
- arrays
 - initializing 807
 - instead of conditional expressions 809
 - processing with macros 807
 - processing with pointers 808
- ASCII
 - decimal character code, mapping for HTML 660
 - non-printable character representation 658
 - text output via conversion to Word 196
- Asian languages, HTML Help support for 331
- aspect ratio, preserving for graphics, HTML 720
- assembling files for distribution 961
 - for Eclipse Help 419
 - for JavaHelp and Oracle Help 379
 - for OmniHelp 369
- assembly directory
 - default files copied to 964
 - emptying before copying to 962
 - files to copy, specifying 962
 - graphics files to copy, listing 966
 - specifying 961
- associative links, *see* **ALinks**
- attribute markers for HTML or XML 834
 - See also* **markers**, **attribute**, for HTML or XML
 - applying 835
 - listed* 835
 - names of 834
 - understanding 834
- attributes, DITA, *see* **DITA**, **attributes**
- attributes, DocBook, *see* **DocBook**, **attributes**
- attributes, HTML
 - align, omitting from paragraph tags 656
 - assigning
 - to character formats 653
 - to paragraph formats 646
 - with attribute markers 835
 - within macro code 845
 - image size, omitting 720
 - image, specifying 718
 - link class, assigning with a marker 610
 - link, assigning with markers 612
 - list, customizing 677
 - table, assigning with markers 737
 - table, automatically generated, eliminating 464, 739
 - WAI, assigning values to 769
- attributes, image, *see* **graphics**, **attributes**
- auto**, FrameMaker command-line option 980
- automating
 - conversions 919, 933
 - Mif2Go** conversions 979
 - production of deliverables 955
- autonumbers
 - converting for database input 944
 - converting to HTML 648
 - converting to Word 160
 - in FrameMaker vs. Word 142
 - including or excluding
 - for DITA XML 484
 - for DocBook XML 562
 - for generic XML 465
 - for HTML 649
 - tabs in, eliminating for HTML 649
- axis, HTML table attribute for WAI
 - guidelines for using 759
 - identifying cells by virtual properties 777
 - purpose and valid content 1013
 - via **Axis** format property 768
 - via **AxisVal** format property 772
 - via **CellAxis** marker 772
 - via [HtmlStyleCellAxis] property 769

B

- background image, for HTML 725
- backslash
 - character literal for macro variables 798
 - escape character
 - in format names 66
 - in frame code for OmniHelp 353
 - in KLink jumps 223
 - in macro code for XML 471
 - in macros 789
 - in RTF code 194
 - in WinHelp font mappings 256
 - separator in file paths

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- for HTML Help 337
 - for OmniHelp 367
 - in code markers for HTML 446
- trailing, to remove line breaks in macros 789, 945
- \$_basefile, macro variable 601, 800
- baseline quotes, converting to straight quotes in macros 790
- \$_basename, system-command variable 800, 939
- \$_basetitle, macro variable 601, 800
- Basic Multilingual Plane (BMP), Unicode 659
- .bat file for system commands 940
- batch conversions via **runfm** 990
- .bct files for WinHelp, location of 1022
- before, macro string operator 818
- beta executables 62
- Bezier curves, in WMF graphics 870
- bgcolor, automatically generated, excluding from HTML table cells 464, 739
- .bha file
 - for HTML Help, location of 1024
 - for JavaHelp, location of 1025
 - for OmniHelp, location of 1024
- .bhc files
 - for Eclipse Help
 - location of 1025
 - for HTML Help
 - location of 1024
 - merging via command line 208
 - for JavaHelp, location of 1024
 - for OmniHelp, location of 1024
- .bhk files
 - for HTML Help
 - location of 1024
 - merging via command line 208
 - for JavaHelp, location of 1024
 - for OmniHelp, location of 1024
- .bhl files for OmniHelp, location of 1024
- .bhm files for JavaHelp, location of 1024
- .bhs files for OmniHelp, location of 1024
- binary index for HTML Help 321
- binary TOC for HTML Help 305
 - for browse buttons 303, 320
 - mid-topic links 323
 - no-link contents entries 322
- bitmaps
 - See also* graphics, bitmap
 - compressing 190, 899
 - embedding in a font, WinHelp 255
 - embedding in Windows metafiles 886
 - reorienting 899
 - rescaling 898
 - resolution of, for RTF 190, 871
 - using different versions of 894
- bitwise operators for macro expressions, *listed* 812
- blank paragraphs, *see* empty paragraphs
- blanks, *see* spaces
- block text, content-model element type 912
- <blockquote>, HTML paragraph tag 647
- blocks for expandable sections
 - configuring 233
 - delimiting
 - with formats 228
 - with markers 229
- blue borders, eliminating from anchored frames 718, 723
- .bmp files, exporting 881
- BMP, Basic Multilingual Plane, Unicode 659
- BMP, graphics export format 130, 884
 - See also* bitmaps
- BMROOT entries in .hbj files 251
- bold, as a format override for HTML 657
- Book Error Log, FrameMaker, for broken links 112
- book file, FrameMaker
 - for HTML conversions 424
 - for RTF conversions 150
- book, **runfm** option 981, 983
- book-level maps, DITA, vs. chapter maps 540
- bookmap, DITA, *see* DITA, bookmap
- bookmarks, Word
 - for cross references 175
 - for every ObjectID 183
 - for interfile cross references 179
 - limiting 148
- bookmeta.xml, DITA <bookmeta> template 549
- books, FrameMaker, converting
 - to HTML 424
 - to RTF 150
 - via **runfm** 983
- books, FrameMaker, nested 53

ABCDEFGHIJKLMNOPQRSTUVWXYZ

border, automatically generated, excluding from
HTML tables 464, 739

borders

around anchored frames, hiding 718, 723
around in-line graphics in Word 900
around table cells, HTML 739, 740
HTML table, colors of 746

branching browse schemes for WinHelp 293

breadcrumb trails

in Eclipse Help 406, 413
in HTML 627

<\$_break>, control structure for macros 815, 817

breaks, line

See also [line breaks](#)
including in DITA via PIs 499
suppressing
 in HTML or XML output 437
 in <pre> text 670
 in XML output 461

breaks, page

See also [page, breaks](#)
and section
 for print RTF 152
 for WinHelp 249
as split points for HTML 587

breaks, topic, including space or a separator in 586

Bristol HyperHelp

format for WinHelp graphics 264
specifying for WinHelp 247

broken links, checking for 112, 1033

browse

buttons, enabling in HTML Help Workshop 303
numbers, WinHelp, specifying 292
prefix, assigning for branches, WinHelp 293
sequences
 HTML, creating 635
 WinHelp, creating 292

browser-dependent

HTML list styles 676
settings for HTML tables 731

browsers

cookies for, created by OmniHelp 354
CSS support in 424, 429, 681, 1001
font rendering differences 666
graphics support in 871

build numbers, finding

in output 1034

on Web site 1035

build numbers, **Mif2Go**, finding
in configuration files 1034

bulleted lists, converting

to DITA XML 487
to DocBook XML 565
to HTML 674
to WinHelp 256
to Word 162
to XML 466

bullets

eliminating from HTML output 649
mapping to special characters for HTML 661
replacing for W3C validity 454
specifying for WinHelp 256

buttons for drop-down links, configuring 232

C

Calibre, for ePub 424

callouts, graphic

for HTML, using FrameMaker export filters 87,
129, 706, 708

for RTF

adjusting 190
adjusting space above 903
font changes in 191
rotated text in 191
suppressing underlines 903
using FrameMaker export filters 873
using Mif2Go graphics processing 128

for WinHelp

adding to replacement graphics 872
avoiding GDI resource leak 264
converting to WMF 263
embedding in WMF 886

CALS table model

default for XML 459
specifying for HTML 730

cancelling a conversion 63

caption, figure 715

<caption> tags for HTML tables, placing 747

cascading style sheets, *see* [CSS](#)

case sensitivity

in FrameMaker vs. Word 142
of ALink keywords 219, 221
of attribute names for HTML links 613

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- of command-line arguments 998
 - of CSS class names 691, 692
 - of file names 947
 - of FileIDs 121
 - of format names, specifying 113
 - of index terms for HTML-based Help 217
 - of key names in configuration settings 104, 922
 - of KLink keywords 219, 223
 - of macro operators 813
- .cdr files, exporting 881
- Cell***, custom markers for HTML table cell
 - attributes 832
- CellAbbr**, custom marker for WAI 772, 1013
- CellAxis**, custom marker for WAI 772, 1013
- CellClass**, custom marker for CSS 739, 832
- CellGroup**, custom marker for WAI 773, 832, 1013
 - using to define a ColGroup cell 767
 - using to define a RowGroup cell 767
- CellID**, custom marker for WAI 772, 832
- cellpadding, automatically generated, excluding
 - from HTML tables 464, 739
- cells, *see* table cells; tables
- CellScope**, custom marker for WAI 772, 832, 1013
- cellspacing, automatically generated, excluding
 - from HTML tables 464, 739
- CellSpan**, custom marker for WAI 773, 781, 832, 1013
- CGM, graphics export format 130, 884
- change bars
 - converted to DITA <chbar> elements 493
 - converting to DITA condition attributes 499
 - converting to RTF 143
 - replacing in HTML or XML 669
- change bars, replacing with tags in HTML/XML 669
- \$\$_chapnum, macro variable 800
- chapter
 - configuration files 855
 - individual, configuration file for 919
 - maps, DITA, vs. book maps 540
 - numbers, for print RTF 181
- char, macro string operator 817
- character
 - See also* characters
 - encoding
 - for code pages 662
 - for HTML 432
 - for HTML Help 296
 - for XML 460
- entity references, HTML 653
- formats
 - coding for WinHelp jumps and pop-ups 281
 - converting to RTF 163
 - mapping to CSS span classes 693
 - properties, overriding 926
 - replacing with code, for WinHelp 257, 822
 - replacing with code, for Word 174, 822
 - to identify Help elements 933
- literals
 - assigning to macro variables 798
 - for macro variables, *listed* 798
- ranges, Unicode, assigning CSS classes to 694
- sets, double-byte 659
- spans, changing properties of 926
- characters
 - See also* character
 - accented, converting to HTML 660
 - double-byte, in HTML 432
 - double-byte, in XML 460
 - high ASCII
 - encoding for HTML 432
 - encoding for WinHelp 255
 - encoding for Word 168
 - encoding for XML 460
 - mapping to HTML 653
 - mapping to RTF 172
 - replacing for W3C validation 454
 - in special fonts, mapping for HTML 662
 - printable set of 658
 - problem, in HTML hypertext links 612
 - representing with a font, WinHelp 255
 - special, avoiding in URIs 663
 - special, converting
 - for Eclipse Help 412
 - for HTML 653
 - for print RTF 172
 - for WinHelp 254
 - special, mapping 660
 - for code pages 662
 - Unicode, FrameMaker 8, encoding for RTF 169
- CharTitle** custom attribute marker for hover text 448
- Chinese
 - for HTML Help output, specifying 332
 - for RTF output, specifying 147
- Mif2Go** support for 53

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- .chm file, compiled HTML Help 296, 336
- CHM files, merging 339
- CHM files, unblocking 296
- Choose Project* dialog, *illustrated* 78
- chrome, browser, for OmniHelp 352
- CJK languages 53, 431
- class, CSS attribute
 - See also* CSS, class names
 - for condition indicators 447
 - for graphics 710
 - for links 610, 833, 1016
 - assigning with a format 611
 - assigning with a marker 610
 - for paragraphs 692
 - for table cells 738
 - for table columns 732
 - for tables 694, 737
 - for Unicode character ranges 694
 - for XML output 462
 - naming restrictions 691
- \$\$_class, macro variable 800
- client, runfm option
 - for Mif2Go 981
 - for other plug-ins 993
- ClientName
 - for other plug-ins 993
- clipping text outside a text frame 902
- close, runfm option 982, 987
 - for remote operation 992
- closing </p> tags, suppressing in HTML 437
- CLSID, Windows Registry key for 992
- CMYK colors converted to RGB
 - for HTML 438
 - problems with 441
 - for WinHelp 258
 - for Word 172
- .cnt file, WinHelp contents 288
 - location of 1022
- code pages
 - encoding for special characters 662
 - for Asian and Cyrillic languages
 - for HTML Help 331
 - for print RTF 147
 - obtaining 54
 - for HTML Help 296
 - omitting, for uncompiled HTML Help 300
- code sections, configuring for HTML 670
- Code, HTML custom marker type 832
- <col> element tags for HTML tables 732
- ColGroup and RowGroup cells 784
 - using with id/headers method 778
 - using with scope attributes 776
- <colgroup> elements
 - and ColGroup cells 785
 - required for scope column groups 776
 - tags for HTML tables 732
- colors
 - CMYK-RGB conversion problems 441
 - defining and mapping for HTML 438
 - for HTML links 610
 - for hypertext links 139
 - FrameMaker default, *listed* 439
 - in graphics
 - for WinHelp 870
 - for Word 869
 - in tables
 - for HTML 745
 - for WinHelp 261
 - for Word 185
 - of condition indicators, displaying in HTML 447
 - text, specifying
 - for HTML 657, 669
 - for print RTF 172
 - for WinHelp 258
 - text, suppressing, for HTML 657
 - Web-safe
 - for HTML tables 746
 - for HTML text 440
 - listed* 440
- colspan, HTML table attribute 760
- column spans in HTML tables 781
- columns, text, specifying for RTF 153
- combining chapter files 73
 - for RTF output 150
- command-line version of Mif2Go
 - dcl.exe
 - examples 1000
 - runfm.exe
 - examples 989
 - syntax 980
- commands, hypertext, remapping 837
- commands, system
 - executing 937

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- in batch files 940
 - in **Mif2Go** macros 940
 - starting, before deletions 957
 - understanding 939
- commenting out configuration sections 107
- comments
 - in configuration files, syntax for 104
 - in macro definitions 789
 - in system-command macros 941
 - substituting for paragraphs in HTML 650
- comparison tool, file, obtaining 60
- compiling
 - Help files for delivery 971
 - HTML Help 333
 - JavaHelp with Helen 384
 - WinHelp 89, 248, 250
- condition indicators, mapping to HTML attributes 447
- condition Show/Hide settings, applying 123
- conditional
 - expressions
 - in macros 815
 - replacing with list variables 809
 - operators for macro expressions, *listed* 813
 - settings, FrameMaker, applying 122
 - text
 - converting to DITA attributes 533
 - converting to HTML attributes 446
 - disallowing for selected DITA elements 534
 - hidden, excluded from output 68
 - HTML tags for 446
 - indicators, displaying in HTML 447
 - mapping to DITA attributes 533
 - Show/Hide settings, applying 68, 123
 - to differentiate output 68, 936
- Config**, custom marker type 832
- configuration
 - file, *see* configuration file
 - macros, *see* configuration macros
 - markers for overrides 921
 - options determined at run time, *listed* 863
 - section, *see* configuration file, sections; configuration sections
 - settings, *see* configuration settings
 - template, *see* configuration templates
 - variables, *see* configuration variables
- configuration file
 - chapter 855
- comment syntax 104
- creating
 - automatically 992
 - for individual chapter files 919
- deciding which to edit 856
- document-specific
 - how created 82
 - name of 60
 - where to keep 854
- editing 91
- location of 62, 1019
- macro, editing 861
- output-specific, editing 860
- project, editing 858
- sections
 - See also* configuration sections
 - names of 103
 - order of 103
- source-specific, editing 859
- structure of 103
- template, *see* configuration templates
- configuration macros
 - accessing settings with 809
 - changing graphics settings with 712
 - changing table settings with 754
 - deploying 810
 - overriding configuration settings with 921
- configuration sections
 - See also* configuration file, sections
 - commenting out 107
 - fixed-key, *listed* 925
 - names of 103
 - order of 103
 - using as list variables 807
 - variable-key
 - cross-reference format, *listed* 928
 - graphic, *listed* 930
 - table format or ID, *listed* 928
 - text format, *listed* 926
- configuration settings
 - case sensitivity of 104, 922
 - changing on the fly 919
 - changing with system commands 940
 - file-path separator in 105
 - fixed-key
 - overriding 924
 - vs. variable-key 104
 - in configuration templates 862
 - order of 104
 - overrides to, persistent vs. temporary 921

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- overriding 920, 931
 - with macros 921
 - with markers 921
- precedence of, *listed* 920
- querying with configuration variables 809
- rules for 102
- spaces and tabs in 104
- syntax of 103
- variable-key, overriding 925
- wildcards in 106, 113
- configuration templates 67
 - See also* templates, configuration 849
 - chaining 863, 919
 - creating 861
 - referencing 851
- configuration variables
 - assigning macros and variables to 923
 - assigning values to 922
 - capturing settings with 809
- Confluence, generating XHTML for 449
- consfile.txt, FrameMaker console output 989
- console messages, FrameMaker
 - for automation commands 956
 - for broken links 112
 - reviewing after **runfm** 988
- constraints, DITA, including 477
- contains, macro string operator 818
- content model
 - See also* content models; DTD
 - abstracting from a DTD
 - for DITA 476, 907
 - for DocBook 907
 - configuration file, producing 907
 - configuration sections 908
 - [ElementSets] 910
 - [Topic] 909
 - [TopicFirst] 910
 - [TopicLevels] 911
 - [TopicParents] 910
 - DITA
 - settings, overriding 914
 - table structure, adding 916
 - topic type, naming 908
 - DocBook, naming 908
 - element levels 911
 - element parents 910
 - element sets 910
 - element types 912
 - first-child elements 910
 - generating from a DTD 906
 - replacing 907, 909
 - root element, specifying 909
- content models
 - See also* content model
 - built-in
 - configurations for, *listed* 906
 - derivation 905
 - obtaining copies of 906
 - debugging 918
 - preparing for use 907
 - working with 905
- content type, XML, specifying 461
- contents
 - converting from FrameMaker TOC 81, 124
 - Eclipse Help
 - creating 411
 - entries, merging from multiple files 415
 - link paths, supplying 412
 - properties, configuring 412
 - properties, specifying 410
 - special characters in 412
 - Help levels, checking 209
 - HTML
 - converting from FrameMaker TOC 444
 - local TOCs 631
 - suppressing page numbers in 445
 - HTML Help
 - and index, generating 319
 - entries, configuring 322
 - links to mid-file topics 323
 - HTML-based Help
 - entries, merging from multiple files 208
 - levels, setting 210
 - lists, maintaining 208
 - JavaHelp
 - creating 385
 - entries, configuring 385
 - entries, merging from multiple files 208
 - expansion levels, specifying 385
 - images, designating 386
 - mid-topic links, avoiding for Help systems 210
 - OmniHelp, including 356
 - WinHelp
 - assembling for multiple topic files 291
 - combined file, specifying 248
 - configuring 288
 - entries, configuring 288
 - level for headings, specifying 248

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- levels, setting 209
 - multiple files, referencing 291
 - referencing secondary windows 291
 - updating an existing file 88
- context IDs for Eclipse Help 418
- context-sensitive Help 239
 - for Eclipse Help, setting up 417
 - for HTML Help, setting up 326
 - for JavaHelp, using symbolic IDs 401
 - for OmniHelp, setting up 364
- <\$_continue>, control structure for macros 815, 817
- controls for macro expressions
 - listed* 815
 - using 815
- conversion
 - DCL modules, writing 1003
 - events, logging 115
 - process, *illustrated* 62
 - project, setting up 78
 - restrictions on file names 51, 65
 - setting up
 - for ASCII DCL 1009
 - for DITA XML 478
 - for DocBook XML 559
 - for Eclipse Help 403, 404
 - for HTML 425
 - for HTML Help 298
 - for JavaHelp or Oracle Help 375
 - for MIF 1006
 - for OmniHelp 345, 346
 - for print RTF 146
 - for WinHelp 244
 - for XML 425, 459
- template
 - for alternate graphics 69
 - for eliminating page numbers from cross references 68
 - for HTML 426
 - for WinHelp 246
 - for Word 152, 155
 - preparing 69
 - specifying at set-up 79
- to HTML
 - preparing for 426
 - validating 431
- to WinHelp
 - basic options 248
 - preparing for 246
- to Word, preparing for 151
- via DCL, preparing for 996
- via **runfm** 979
- converting
 - See also*
 - conversion
 - converting to HTML
 - converting to RTF
 - converting via DCL
 - converting via **runfm**
 - FrameMaker documents, steps for 82
 - FrameMaker-generated files 124
 - multiple projects via **runfm** 990
 - single file of a book 937
 - system variables to text 114
 - for HTML 437
 - for RTF 157
 - to ASCII DCL 1009
- converting to HTML
 - contents entries 444
 - cross references 617
 - equations 725
 - footnotes 671
 - FrameMaker books 424
 - generated files 441
 - graphics 703
 - hypertext links 619
 - index entries 442
 - list formats 674
 - nested lists 676
 - numbered lists 676
 - run-in paragraph formats 648
 - smart quotes to straight quotes in macros 790
 - special characters 653
 - table footnotes 748
 - tables 727, 754
 - via command line, example 1001
- converting to RTF
 - character formats 163
 - columns 153
 - cross references 174
 - EPSI graphics 875
 - footnotes 153
 - graphics 186
 - headers and footers 154
 - index entries for Word 181
 - paragraph anchors 171
 - reference frames 162
 - run-in headings 160
 - screen shots and other bitmaps 899

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- sidehead formats 159
- special text flows 156
- tab settings 163
- tables 184
- via command line, example 1001
- converting via DCL
 - ASCII DCL to binary DCL 1001
 - in multiple steps, example 1002
 - single file, example 996
 - to HTML, example 1001
 - to RTF, example 1001
- converting via **runfm** 979
- cookies, OmniHelp, persistence of 354, 371
- copyright statement and date, for WinHelp 250
- `$$_count`, macro variable 800, 817
- crash, debugging 1035
- cropped WMF graphics in WinHelp 870
- cross references
 - See also* cross-reference
 - broken
 - checking for 112
 - resolving with a second pass 1028
 - eliminating page numbers from 68
 - for DITA XML
 - converted to `<xref>` elements 527
 - format attribute, specifying 529
 - in `.ref` files 1023
 - links below topic level 528
 - omitting from footnotes 529
 - retaining content of 528
 - scope attribute, specifying 529
 - type attribute, specifying 530
 - understanding how converted 527
 - wrapper outputclass, specifying 528
 - for DocBook XML
 - converted to `<xref>` elements 569
 - in `.ref` files 1023
 - for HTML
 - converting to links 617
 - converting to text 618
 - deleting 618
 - in `.ref` files 1019, 1023
 - omitting paths from 622
 - for print RTF 174
 - external, creating 179
 - external, enabling 179
 - from master pages, configuring 181
 - locking 175
 - omitting from output 177
 - for WinHelp
 - converting 259
 - converting to text 260
 - deleting 260
 - forward, resolving 790, 1028
 - FrameMaker, IDs for 117
 - in macros 790
- cross-file links, *see* interfile links
- cross-reference
 - See also* cross references
 - identifiers, FrameMaker and **Mif2Go** 117
 - jump destinations, WinHelp, specifying 260
 - links
 - to text insets for HTML 624
 - links, in DITA XML 527
 - marker text, truncating in WinHelp 261
 - markers
 - duplicate, resolving 118
 - Word-generated, eliminating 114
 - properties, overriding 928
- CSH, *see* context-sensitive Help
- CSS 681, 701
 - browsers, supporting 424, 1001
 - class attributes for table columns 732, 733
 - class names
 - case of 692
 - for character formats 693
 - for footnotes 694
 - for links, assigning with formats 611
 - for links, assigning with markers 610
 - for paragraph formats 692
 - for table cells 738
 - for table footnotes 694
 - for tables 694, 737
 - for XML tags 462
 - restrictions on 691
 - directory to copy files from, specifying 969
 - file name, specifying 686
 - file, specifying when to create 683
 - files
 - for OmniHelp 350
 - list of, to copy 970
 - font sizes, mapping 664
 - font-size units, changing 699
 - for OmniHelp navigation panel, modifying 355
 - line leading in 700
 - options for OmniHelp 350
 - properties, specifying 691

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- span class for character formats 693
 - using with HTML Help 303
 - vs. HTML, header formatting 429
 - .css file, cascading style sheet (CSS) 1001
 - location of 1023, 1024
 - options, specifying 683
 - curly quotes, converting to straight quotes
 - for print RTF 172
 - for WinHelp 256
 - in macros 790
 - \$\$_currbase, macro variable 601, 800
 - \$\$_currfile, macro variable 601, 800
 - \$\$_currfilepath, macro variable 601, 800
 - \$\$_currpath, system-command variable 800, 939
 - \$\$_currtitle, macro variable 601, 800
 - custom markers
 - See also* markers, custom
 - for DITA maps 556
 - for DITA XML 536
 - for DocBook XML 582
 - for HTML extracts 602
 - for WAI
 - advantages of 760
 - attributes 756
 - image attributes 757, 1014
 - link attributes 759, 1016
 - table attributes 762, 1013
 - table cell attributes 772
 - custom ruling and shading in tables, for HTML 727
 - CVS revision management, generating MIF output
 - for 1006
 - Cyrillic
 - font encoding
 - for HTML/XML 432
 - for WinHelp 255
 - for Word 168
 - FrameMaker 8 Unicode, for Word 169
 - languages, HTML Help support for 331
 - locale, for index sort order 218
 - Czech
 - for HTML Help output, specifying 332
 - for RTF output, specifying 147
- ## D
- Darwin Information Typing Architecture, *see* DITA
 - dashed lines, in WMF graphics 870
 - Data Type Definition, *see* DTD
 - database input from HTML 450
 - dcb, DCL output type 1000
 - .dcb, output file extension for binary DCL files 62, 111, 1001, 1009, 1011
 - dcl, DCL output type 1000
 - DCL, **Mif2Go** command-line version
 - advantages of 992
 - compared with **runfm** 991
 - conversion modules, writing 1003
 - converting between ASCII and binary output 1001
 - files, using existing 85, 111
 - intermediate conversion files 62
 - options, *listed* 1000
 - output
 - ASCII, converting to 111, 131
 - file extension, specifying 427
 - file structure 1003
 - files and paths, specifying 1002
 - from MIF via command line 1002
 - .dcl, output file extension for ASCII DCL files
 - convert existing DCL files 85, 111
 - convert from one form of DCL to another 1001
 - convert to ASCII DCL 1009, 1011
 - specifying via *Export* dialog 85
 - dcl.exe, DCL executable 1017
 - DCOM, enabling for remote operation 992
 - \$\$_dcount, macro variable 800, 817
 - dcx.dll, DCL ASCII-binary converter 1002, 1017
 - debugging options 116, 1035
 - default configuration values 102
 - definition lists, DITA
 - first-child status 505
 - Delete**, HTML custom marker type 832
 - deleting
 - files before conversion 958
 - old MIF files 960
 - deliverables
 - assembling for distribution 961
 - compiling or archiving 971
 - file names and extensions, *listed* 975
 - producing 956
 - destinations, named, from Word cross-reference markers 114
 - diag, **runfm** option 982, 988

ABCDEFGHIJKLMNOPQRSTUVWXYZ

DIB, *see* [bitmaps](#)

dictionary lists, converting
to HTML [677](#)
to HTML tables [824](#)

directory
Omni Systems home, creating [54](#)

directory names
restrictions on [51](#)

directory names, restrictions on [65](#)

distribution
assembling files for [961](#)
files, *listed* [1017](#)
Mif2Go, downloading [56](#)

DITA

attributes
assigning to elements [495](#)
block element, overriding [497](#)
block element, specifying [497](#)
class, not included [477](#)
collection-type [547](#)
for bookmark wrapper elements [555](#)
from FrameMaker conditions [533](#)
href, omitting topic IDs from for
FrameMaker 8 [543](#)
ID, specifying [495](#)
image width and height [519](#)
index range, for DITA 1.1 only [480](#), [500](#)
inline element, specifying [498](#)
omitted from reopened paragraph tags [484](#)
outputclass
assigning [498](#)
for CSS [485](#)
parent, interpolated, assigning [498](#)
parent, interpolated, overriding [498](#)
root element, specifying [497](#)
table type, assigning [511](#)
<xref>, overriding [529](#)
bookmark
<bookmeta> information [549](#)
<booktitle> information [548](#)
constructing [548](#)
declarations, overriding [915](#)
mapping FrameMaker files to [551](#)
multiple files per role [554](#)
multiple roles per file [554](#)
retable, book level, excluding [550](#)
roles for chapter files [551](#)
structure, specifying [550](#)
type of, specifying [548](#)

wrapper element attributes [555](#)

configuration file, custom topic type
creating [913](#)
listing [915](#)
locating [916](#)

content models

See also [content model](#); [DTD](#)
abstracting from DTD [476](#)
built-in, configurations for, *listed* [906](#)
built-in, obtaining copies of [906](#)
built-in, source of [905](#)
configuration files, producing [907](#)
configuration sections [908](#)
generating from DTDs [906](#)
naming [908](#)
overriding [914](#)
preparing for use [907](#)

DTD

properties, specifying [476](#)
SYSTEM identifier, configuring [481](#)

elements

assigning to formats [487](#)
block, ID, specifying [495](#)
block, nesting [501](#)
default, for character formats [494](#)
default, for paragraph formats [489](#)
delimiting [486](#)
footnote [484](#)
graphic, alternate text for [518](#), [896](#)
graphic, including [518](#)
image, configuring [516](#)
levels, overriding [510](#)
levels, specifying [509](#)
list types, parents of [502](#)
<menucascade> [494](#)
nesting [477](#)
outputclass attribute [495](#), [498](#)
overriding character mapping [494](#)
overriding paragraph mapping [490](#)
possible parents of, specifying [502](#)
root, assigning outputclass attribute [499](#)
<shortdesc> [500](#)
typographic [494](#)
<uicontrol> [494](#)

equations as <image> elements [136](#)

images

alternate text for [518](#)
ancestry, specifying [516](#)
configuring [516](#)
DPI values for [518](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- parents of 516
 - width and height attributes 519
 - wrapping in <fig> elements 517
- importing into FrameMaker 8 481
- maps 915
 - book maps vs. chapter maps 540
 - from FrameMaker chapters and books 477
 - ID, specifying 542
 - levels, specifying 544
 - naming 541
 - navigation aids, providing 547
 - navigation title, specifying 542
 - nesting 540
 - overriding settings with markers 556
 - overwriting 540
 - predefined markers for, *listed* 556
 - titles, specifying 541
- marker types, predefined
 - for maps, *listed* 556
 - for topics and elements, *listed* 536
- migrating legacy data to 474
- parent elements, specifying 502
- producing 473, 539
- project, setting up 478
- PUBLIC declaration 483
- relationship tables
 - adding ALink rows to 546
 - book level, excluding 550
 - collection-type attribute 547
 - excluding ALink column from 546
 - structure of 546
 - unidirectional linking in 547
- resources 474
- specializations 477
- specialized topic types
 - debugging 918
 - element levels 911
 - element parents, specifying 910
 - first-child elements 910
- table structure, specialized 916
 - mapping from table type 916
 - properties, assigning 917
 - undefining 917
- tables
 - ancestry, specifying 512
 - as image containers 514
 - attributes, assigning 511
 - cell properties 514
 - column widths of, specifying 513
 - converting from FrameMaker 510
 - default table type 511
 - empty paragraphs in 513
 - mapping formats to types 511
 - omitting code for 515
 - omitting element ancestries 513
 - parents of 512
 - titles of, converting 514
 - width, specifying 513
- text insets, delimiting 534
- topic files, splitting 520
- topic ID, specifying 526
- topic types
 - assigned via marker 525
 - assigned via paragraph format 525
 - assignment, precedence of 524
 - default, specifying 525
 - predefined, overriding 914
 - specializing 913
 - specifying 524
- topics
 - alternate titles for 526
 - embedded 519, 521
 - IDs, specifying 526
 - map levels, specifying 544
 - nesting 521
 - organizing 519
 - starting paragraph 523
 - wrapping in <dita> elements 521
 - version, specifying 480
- DITA Open Toolkit 475
- DITA*, DITA predefined marker types 832
 - for maps, *listed* 556
 - for topics and elements, *listed* 536
- DITA-FMx plug-in 519
 - and book-level ditamaps 541
 - compatibility with 481
 - fmdpi attribute, value for 518
 - recommended 480
- DITA-FMx plug-in, Leximation 543
- .ditamap, DITA map file 539
- \$\$_ditastart, macro variable 491, 800
- DLL files 62
 - build numbers of 1035
 - downloading 62
 - listed* 1017
- .doc files, for Word 2000 192
- doc, runfm option 981, 983
- DocBook

ABCDEFGHIJKLMNOPQRSTUVWXYZ

attributes

- ID, assigning 569
- inline, assigning 572
- other than ID, assigning 571
- overriding 572
- parent, assigning 572
- parent, overriding 572

content model

See also [content models](#); [DTD](#)

built-in

- configurations for, *listed* 906
- obtaining copy of 906
- source of 905

configuration file, producing 907

configuration sections 908

debugging 918

generating from a DTD 906

naming 908

preparing for use 907

elements

- assigning to formats 565
- block, ID, specifying 571
- block, nesting 573
- default, for character formats 569
- default, for paragraph formats 566
- figure, options for 581
- levels, overriding 580
- levels, specifying 579
- list types, parents of 574
- overriding character mapping 569
- overriding paragraph mapping 567
- possible parents of, specifying 573

images

- ancestry, specifying 581
- figure element, what to include in 581
- omitting size attributes from 582
- options for, specifying 581
- parents of 581
- titles, where to place 581

language attribute, specifying 431

marker types, predefined, *listed* 583

migrating legacy data to 558

output, producing 557

parent elements, specifying 573

resources 557

tables

- ancestry, specifying 580
- parents of 580

DocBook*, DocBook predefined marker types 832
listed 583

DocType, specifying for HTML/XML 429

document

- FrameMaker, converting via **runfm** 983
- information file, creating 1001
- layout options, specifying for RTF 151
- properties, importing 865
- properties, specifying for HTML 436

Document Coding Language, *see* [DCL](#)

double-byte characters

- in HTML 432
- in XML 460
- sets of 659

double-byte languages 53, 431

downloading 62

- beta executables 62
- evaluation version 1029
- HTML Help Workshop 58, 297
- JavaHelp 58
- Microsoft Help Workshop 58, 243
- OmniHelp control files 342
- run-time libraries 62
- User's Guide 41

DPI

- default, specifying for HTML 436
- of equations
 - specifying 136
 - specifying for HTML 884
- of images
 - including in DITA image attributes 518
 - specifying, for HTML 884
 - understanding 130

drmf.dll, DCL reader for MIF files 1017

drop-down sections

See also [expandable sections for HTML](#)

blocks for

- configuring 233
- delimiting with formats 228
- delimiting with markers 229

CSS for 233

emulating Web Works Publisher method 237

JavaScript code for 234

links for

- configuring 230
- delimiting with formats 228
- delimiting with markers 229

DTD

See also [content model](#)

abstracting content model from 906

ABCDEFGHIJKLMNOPQRSTUVWXYZ

DITA, properties, specifying 476
 HTML, specifying 429
 parameter entities, equivalent to element sets 910
 source for built-in content model 905

dttd2ini, content-model extractor 476, 906
dwhtml.dll, DCL writer for HTML files 1017
dwinf.dll, DCL writer for .inf files 1017
dwrtof.dll, DCL writer for RTF files 1017
 dynamic Help systems, *see* modular Help systems

E

eBooks, producing 424
 from HTML 58

Eclipse Help

contents and index methods 411
 contents properties, configuring 412
 context file, naming 417
 context-sensitive Help, setting up 417
 files, packaging 419
 generating 403
 index properties, configuring 414
 infopops, configuring 417

MANIFEST.MF

configuring 408
 including or excluding 407

output options, specifying 405

plug-in

CSH properties, specifying 411
 ID, specifying 408
 index properties, specifying 410
 naming 407
 product version, specifying 408
 provider, specifying 408
 schema version, specifying 410
 TOC properties, specifying 410

plugin.xml

configuring 409
 creating 409
 excluding 406

projects

merging 415
 setting up 403
 TOCs, primary vs. secondary 415
 understanding 403

Eclipse SDK, downloading 58

EclipseAnchor, custom marker type 416, 832

EclipseContext, custom marker type 418, 832

EclipseLink, custom marker type 416, 832

EDD, for exporting structured documents 135

editor

for log file error display, designating 115

eHelp 200

electronic books, producing 424

\$\$_element, macro variable 800

elements

DITA, configuring 486

DocBook, configuring 564

structured FrameMaker, formats required for 135

XML, from unstructured text 462

<\$_else>, control structure for macros 815

<\$_elseif>, control structure for macros 815

embedded graphics 877

See also graphics, embedded

exporting 69, 879, 1010, 1012

from Word, extracting 886

replacing with referenced graphics 69

embedded topics, DITA 519, 521

IDs for, generated 526

embedding bitmaps

in a font, WinHelp 255

in Windows metafiles 886

empty paragraphs

avoiding splits on, for HTML 587

eliminating for HTML output 652

in DITA table cells, retaining tags for 513

in DocBook table cells, retaining tags for 564

in HTML table cells

omitting tags for 744

providing content for 744

retaining tags for 744

in HTML text, providing content for 651

in RTF output, removing final 174

<\$_endif>, control structure for macros 815, 816

<\$_endrange> marker, *see* index, ranges

<\$_endrepeat>, control structure for macros 815

ends, macro string operator 818

<\$_endwhile>, control structure for macros 815

entity references

for HTML 429

for XML 460

mapped from high ASCII characters 653

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- environment variable %OMSYSHOME%, creating 55
- .eps files, exporting 875, 881
- EPS graphics
 - converting 875
 - export format 130, 884
 - replacing 876
- ePub format, producing from XHTML 424
- ePub, producing
 - from HTML output 52, 58
 - from XHTML 424
- equations
 - converting to DITA 518
 - converting to HTML 725
 - converting to Word 143
 - DPI and size, specifying 136
 - for HTML 884
 - exporting 129, 136, 883
 - MathFullForm, including in DITA alt tags 518
 - output format, specifying 137
 - positioning in RTF 137
 - understanding how processed 136
- error messages
 - Could not run DCL filter 1031
 - DCL NT console driver 1035
 - Eclipse Help packaging, viewing 420
 - HTML Help
 - alias entries 331
 - compilation, viewing 334
 - page cannot be displayed 301
 - JavaHelp search index, viewing 388
 - logged as conversion events 115
 - Mif2Go**
 - arguments unacceptable 1031
 - error processing m2rbook command 1030
 - file not found 1031
 - OmniHelp Loading... 1031
 - Oracle Help search index, viewing 390
 - runfm, reviewing 988
 - suppressed, during runfm 989
 - system command, displayed 939
 - text of, localizing for OmniHelp 354
 - WinHelp
 - compilation, viewing 251
 - unmatched braces 256
 - Word
 - cannot open file 177
 - missing formats, eliminating 156, 163
- errors
 - corrupt file, fixing via MIF 1032
 - corrupt graphics 129
 - duplicate keys in configuration settings 103
 - link, checking for 112
 - logged to conversion log file 115
 - severity level of 116
 - WinHelp compiler 212
 - too many tabs 253
 - WinHelp memory 251
 - Word font type 167
- escape character for macros 789
- evaluation version 1029
- evaluation version of **Mif2Go**
 - using to wash files via MIF 1032
- event log, *see* log file
- events, conversion, logging 115
- expandable sections for HTML 226
 - See also* drop-down sections
 - delimiting with formats 228
 - delimiting with markers 229
 - JavaScript code for
 - deploying 234
 - locating 234
 - modifying 235
 - JavaScript macro for, naming 234
 - understanding 227
- Export dialog, skipping 112
- Export*, [GraphExport] keywords:
 - ExportOleFiles 882
 - ExportWmfFiles 881, 882
- exported graphics files, naming 69, 134, 1010
- exporting
 - embedded graphics 69, 879, 1010, 1012
 - HTML for database input 450
 - structured documents 135
- expressions, macro
 - conditional, in macros 815
 - results of 811
 - displaying in output 813
 - using indirection in 819
 - using list variables in 818
- ExtCode***, custom marker types 832
- extension point, Eclipse Help 403, 411, 420
- extension, *see* file, extension
- external vs. internal metafiles, specifying 873
- Extr***, custom marker types 833
 - listed* 602

ABCDEFGHIJKLMNOPQRSTUVWXYZ

`$$_extr*`, predefined macro variables for extracts,
listed 603

extracts, HTML 591
 customizing 601
 delimiting 592
 with existing formats 592
 with markers 593
 with special formats 592
 enabling and disabling 591
 meta text for 598
 naming, with custom markers 947
 referencing 600
 replacing with links 603
 titles of, specifying 594

`<$_extrthumb>`, predefined macro 603, 606

`exwmf.exe`, embedded graphics extractor 1017

F

FAR, for Microsoft Help Viewer 200

favorites option
 for HTML Help 304
 for JavaHelp2 383

`fcharset` values for non-Western scripts 168

FDK name, **Mif2Go** FileID+ObjectID identifier
118

figures, list of (LOF)
 See also [list of figures or tables, converting](#)
 converting to HTML 444
 converting to HTML Help 325
 converting to Word 181
 preventing conversion of 125

figures, *see* [graphics](#)

file

See also [files](#)
 comparison tool, obtaining 60
 corruption, fixing via MIF 1032
 extension
 for DITA XML output 480
 for DocBook XML output 561
 for graphics, HTML 888
 for HTML/XML/DCL output 427
 for interfile links 427
 for Word interfile links 180
 for Word output 147
 for XML output 460
 extracts, HTML, creating 591, 608
 FrameMaker book

 for HTML conversions 424

 for RTF conversions 150

identifiers, *see* [FileIDs](#)

names

 containing blanks 999, 1001
 for DITA topics, via FrameScript 480, 526
 for Word interfile references 180
 HTML project 1026
 HTML split and extract 593
 HTML split and extract, customizing 952
 HTML, custom markers for 947
 of chapter-specific configuration files 919
 restrictions on 51, 52, 65

path, *see* [path](#)

paths in configuration settings 105

sequence, specifying for HTML 644

structure, DCL 1003

titles, HTML, specifying 433

FileID configuration file 120, 623, 1026, 1027
 location of 62

`$$_fileid`, macro variable 800, 952

FileIDs

See also [mif2go.ini](#); configuration file

 for HTML

 determining 717

 finding 729

 modifying 121

 project 120, 623, 1027

 providing for batch conversions via DCL 997

FileName, custom marker type 833, 947

 for DITA topics 526

files

See also [file](#)

 configuration, *see* [configuration file](#)

 converted

 default location of for Word 181

 writing to a different directory 1001

 converting

See also [converting](#)

 individual chapters 937

 to HTML via command line 1001

 to RTF via command line 1001

 copying via system commands 937

 corrupt, washing via MIF 1032

 DCL, using existing 85

 deliverable

 assembling for distribution 961

 default base names of, *listed* 975

 distribution, *listed* 1017

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- generated
 - converting to HTML Help 325
 - converting to WinHelp 265
 - converting to Word 181
 - including in output 81, 124
- graphics
 - See also* graphics, files
 - copied for postprocessing, *listed* 966
 - extension, specifying for HTML 888
 - identifying 133
 - original names of, keeping for HTML 889
 - path, removing for HTML 704, 887
 - replacing, renaming, relocating for HTML 887
- Help contents
 - for HTML Help, generating 319
 - for JavaHelp, creating 385
 - for WinHelp, assembling for multiple topics 291
 - for WinHelp, naming 288
- HTML
 - extracting 591
 - importing as insets 446
 - renaming, for automated systems 946
 - split and extract, referencing 600
 - split and extract, renaming 946
 - split and extract, specifying titles for 594
 - splitting 586
 - splitting, at table heads 587
- index
 - for HTML Help, generating 319
 - for JavaHelp, generating 385
- JavaHelp
 - helpset, configuring 382
 - merging multiple contents and index 208
- macro, individual 793
- macro, library 794
- map, HTML Help, specifying 336
- MIF
 - existing, using 85
 - intermediate, deleting 85
 - managing 111
 - old, deleting before converting 960
- Mif2Go
 - moving 63, 1025, 1026, 1027
 - project, location of 1019
 - naming, restrictions on 51, 65
 - output, copied for postprocessing, *listed* 964
 - postprocessing via system commands 937
 - renaming via system commands 937
 - RTF, conversion 1022
 - saving as MIF 1006
 - splitting
 - for DITA 520
 - for HTML 586
 - WinHelp
 - multiple, referencing from contents 291
 - project, naming 248
- filters, input and output 62
- find, *see* full-text search
- Finished* dialog, skipping 112
- Firefox new window option for OmniHelp 372
- first, macro string operator 817
- \$\$_firstfile, macro variable 601, 800
- \$\$_firstsfile, macro variable (*deprecated*) 601
- fixed-key configuration sections
 - listed* 925
 - overriding settings in 924
 - vs. variable-key 104
- fixed-text links for expandable sections 229
 - configuring 232
- flags for conditions in HTML 447
- flows, text
 - converting to RTF 156
 - including in HTML 113
- folder, *see* directory
- font
 - See also* fonts
 - default, specifying for RTF 166
 - encoding for FrameMaker 8 Unicode, for RTF 169
 - encoding for non-Western characters
 - for WinHelp 255
 - for Word 168
 - graphics default, specifying 902
 - ignored for high ASCII character mapping 662
 - metrics, specifying for RTF 165
 - size units in CSS 303, 699
 - size, changing in HTML Help 303
 - size, matching, for graphics 901
 - tags, HTML
 - including in HTML output 665
 - removed for CSS 688
 - workaround for browser differences 666
 - using to embed bitmaps, WinHelp 255
 - using to represent a character, WinHelp 255
- fonts

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- See also* [font](#)
 - encoding for WinHelp [255](#)
 - mapping, for HTML [663](#)
 - non-Western, encoding
 - for WinHelp [255](#)
 - for Word [168](#)
 - OpenType and TrueType, browser support for [666](#)
 - referenced, in WMF graphics [870](#)
 - special, mapping characters in [662](#)
 - unused, removing
 - for RTF [170](#)
 - for WinHelp [257](#)
- footer rows of tables
 - identifying, HTML [732](#)
 - in HTML `<tfoot>` elements [776](#)
 - positioning for HTML [733](#)
- footers
 - aligning with graphics for print RTF [155](#)
 - and headers, converting to Word [154](#)
 - and headers, *not* converted to HTML [645](#)
- footnotes [671](#)
 - ALink, adding to WinHelp topics [285](#)
 - converting
 - to HTML [671](#), [694](#)
 - to print RTF [153](#)
 - to WinHelp [258](#)
 - inline, configuring, for HTML/XML [672](#)
 - jump, formatting with macros [673](#)
 - links to, eliminating [672](#)
 - omitting
 - from DITA XML output [484](#)
 - from HTML/XML output [671](#)
 - separator for [671](#)
 - table
 - converting to HTML [694](#), [748](#)
 - positioning in HTML [748](#)
 - using list tags vs. `<div>` and `<p>` tags [672](#)
- forced returns
 - converted to spaces
 - for DITA XML [476](#)
 - for DocBook XML [559](#)
 - converting, for HTML or XHTML [651](#)
 - eliminating
 - for DITA [484](#)
 - for DocBook [562](#)
 - for generic XML [465](#)
 - in `<pre>` text, omitting line breaks for [670](#)
- format strings in macro expressions [813](#)
- formats
 - See also* [character, formats](#); [paragraph, formats](#)
 - character, properties of, overriding [926](#)
 - for exporting graphics, HTML [884](#)
 - importing from a conversion template [863](#), [936](#)
 - list, converting to HTML [674](#)
 - mapping
 - to DITA XML [486](#)
 - to DocBook XML [564](#)
 - to generic XML [462](#)
 - to HTML [645](#), [679](#)
 - naming [66](#)
 - paragraph
 - See also* [paragraph, formats](#)
 - deleting, for WinHelp [253](#)
 - merging, RTF [159](#)
 - properties of, overriding [926](#)
 - replacing with code, for WinHelp [257](#), [822](#)
 - replacing with code, for Word [174](#), [822](#)
 - script, designating for HTML [650](#)
 - suppressing, for WinHelp [253](#)
 - paragraph, converting
 - run-in, to HTML [648](#)
 - sidehead, to print RTF [159](#)
 - sidehead, to WinHelp [252](#)
 - unused, removing
 - for print RTF [163](#)
 - for WinHelp [257](#)
- forward cross references, resolving [790](#), [1028](#)
- Frame Vector facets
 - as imported WMF graphics [870](#)
- FrameID, FrameMaker ObjectID for graphics [118](#)
- FrameImage, EPSI preview facet [875](#)
- FrameMaker
 - book file, role in conversion [150](#), [424](#)
 - condition settings, applied at run time [122](#)
 - console messages, reviewing after **runfm** [988](#)
 - export filters, converting graphics with [884](#)
 - generated files, including or excluding [81](#), [124](#)
 - ObjectIDs [118](#)
 - pen style patterns, mapping [900](#)
 - setting up for unattended operation [980](#)
 - templates, applying [863](#)
 - user variables
 - as **Mif2Go** macro variables [802](#)
 - values supplied at run time [123](#)
 - version 8
 - DITA import compatibility [481](#)
 - saving files as version 8 MIF [1008](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Unicode font encoding for RTF 169
 version9, support for 53

FrameMaker.M2G, default **runfm -progid** value 982

frames

- anchored
 - See also* [anchored frames](#)
 - hiding borders of 718, 723
 - object attributes of, specifying 896
 - positioning for RTF 191, 264
 - tables in, converting to RTF 128
- reference
 - converting to RTF 162
 - converting to WinHelp 253
 - removing for HTML 651
- unanchored
 - See also* [unanchored frames](#)
 - excluding from HTML output 713
 - in HTML 886
 - on body pages, for HTML 886
 - on master pages, for RTF 885

framesets

- image maps in 725
- in HTML 450
- in HTML Help 452
- in OmniHelp, customizing 352
- target for HTML jumps 725

framework for Omni Systems applications 54

FTS, *see* [full-text search](#)

full-text search

- for HTML Help 326
 - excluding topics from 326
- for JavaHelp and Oracle Help 387
- for OmniHelp 356, 361
 - excluding content from 363
 - excluding stop words from 363

G

GDI resource leak, WMF graphics problem 74, 264

generated files

- converting 124
 - to HTML 441
 - to HTML Help 325
 - to WinHelp 265
 - to Word 181
- excluding from output 125
- including 124

- at set-up time 81
- in DCL output 1011
- in MIF output 1007

generating book before converting 126

generator, HTML, specifying 433

GhostScript, PostScript interpreter 876

- for converting EPSI graphics 131

.gif files, exporting 881

GIF, graphics export format 130, 884

glossary, converting to JavaHelp 392

gotolink markers

- converting to HTML links 609, 619, 620
- for HTML image maps 723
- for WinHelp jumps and pop-ups 276
- inserting 935

gotopage markers, converting to HTML links 619

Graph*, custom markers for HTML image attributes 833

GraphAlt, custom marker for WAI image attribute 757, 1015

\$\$_graphbase, macro variable 711, 800

GraphDpi, custom marker for image resolution 722, 833

graphic

- See also* [graphics](#)
- elements, including in DITA output 518
- ID, determining for HTML 717
- text, background, specifying 903

Graphic Workshop 130, 134, 873, 874

graphics

- See also* [graphic](#)
- adding space before, HTML 717
- aligning headers and footers with, RTF 155
- aligning, for HTML 714
- alternate text for
 - DITA 518
 - HTML 719
 - WAI 757
- alternate, via conditional text 69
- aspect ratio, preserving, HTML 720
- attributes
 - See also* [image attributes](#)
 - size, omitting 720
 - specifying 718
 - width and height units 722
- bitmap
 - See also* [bitmaps](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- compressing 899
- embedding in Windows metafiles 886
- generating via **Mif2Go** plug-in 992
- borders around
 - eliminating for HTML 718, 723
 - positioning for Word 900
- captions, aligning for HTML 715
- class, assigning 710
- configuration sections subject to override, *listed* 930
- converting
 - for HTML/XML 703
 - for print RTF 186
 - for WinHelp 263
 - in unanchored frames, HTML 713, 886
 - with FrameMaker export filters 129, 884
 - with Graphic Workshop 134
 - with **Mif2Go**, for RTF 128
 - with third-party tools 130
- directory
 - emptying before copying files to 967
 - specifying for assembly 968
 - specifying for HTML links 887
- DPI
 - including in DITA image attributes 518
 - specifying for HTML 884
- embedded 69, 877
 - exporting 111, 131, 1010
 - exporting before converting 69, 879
 - exporting via ASCII DCL 1012
 - imported from Word, extracting 886
- EPSI, replacing 876
- excluding
 - from HTML output 713
 - from RTF output 895
- exporting 871
 - before converting 132
 - embedded 69, 132, 877
 - from master pages, with FrameMaker filters 885
- file extension, specifying, HTML 888
- files
 - assembling for distribution 965
 - copying to assembly directory 965
 - exported, naming 69, 134, 1010
 - for assembly, listing 966
 - identifying 133
 - keeping original names of, HTML 889
 - path to, on UNIX server 705
 - removing path from, HTML 704, 887
 - renaming extension, for Bristol Hyperhelp 247
 - replacing, renaming, relocating, for HTML 887
 - spaces in names, replacing, HTML 889
 - thumbnail, naming 604
- font, specifying default 902
- formats, HTML, preferred 871
- fuzzy, correcting for HTML 703
- GDI resource leak with WMF 264
- groups
 - assigning properties to 708
 - creating with overrides 930
 - creating, HTML 708, 710
- imported
 - by copying, *see* **graphics**, *embedded*
 - by reference, replacing 890
 - from Word, extracting 886
- in extracts, referencing 607
- in table cells, repositioning
 - for HTML 717
 - for RTF 185
- including without converting, HTML 889
- indenting, HTML 716
- JavaHelp, specifying location of 380, 394
- macros, specifying, HTML 715
- master page
 - in unanchored frames, for RTF 885
 - including for HTML 885
 - including for Word 154
- names of, including in Word 192
- omitting from HTML or XML output 708
- paragraph format, specifying
 - for WinHelp 264
 - for Word 191
- properties
 - accessing with <\$\$_extrgraphid> 607
 - overriding for HTML 895, 929
 - overriding for RTF 895
 - specifying, HTML 718
- reference page, skipping, HTML 885
- replacement, format for, HTML 706, 888
- scale, preserving in Word 191
- scaling
 - for HTML/XML 719
 - for WinHelp 872
 - for Word 191
- settings
 - custom, specifying 895
 - overriding 896

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- synchronizing for HTML output 968
- synchronizing for RTF output 969
- size of, preserving, for Word 191
- spacer, for HTML 716, 747
- tags around, omitting 713
- thumbnails, referencing, HTML 604
- visible portion only, converting 712
- WAI markup for 756
- writing without converting text 88
- GraphLongdesc**, custom marker for WAI image attribute 757, 1015
- \$\$_graphorighigh, macro variable 711, 800
- \$\$_graphorigwide, macro variable 711, 800
- \$\$_graphsrc, macro variable 711, 800
- GraphTitle**, custom marker for WAI image attribute 757, 1015
- Greek
 - for HTML Help output, specifying 332
 - for RTF output, specifying 147
 - Mif2Go support for 53
- Greek font encoding for HTML/XML 431
- groups
 - graphic, *see* [graphics, groups](#)
 - table, *see* [tables: HTML, groups](#)
- .grx file, graphics references
 - deleting between conversions, HTML 427
 - location of 1019, 1022
 - use for export 88
- guillemets, converting to straight quotes in macros 790

H

- <h1> - <h6>, HTML paragraph tags 647
- H2reg, for Microsoft Help Viewer 200
- hard returns
 - ignored for preformatted HTML elements 670
 - in configuration overrides 931
 - See also* [forced returns](#)
 - to end WinHelp topic titles 272
- line breaks
- headers
 - aligning with graphics, RTF 155
 - and footers
 - adjusting for Word 154
 - converting to RTF 154
 - not converted to HTML 645
 - running, for RTF 156
 - levels of, for WinHelp 289
- headers, HTML table attribute for WAI 759, 1013
 - purpose of 763
- heading rows in tables, identifying, HTML 732
- headings, run-in, converting
 - to RTF 160
 - to WinHelp 252
- Helen, third-party JavaHelp compiler 384
- Help 2, Microsoft, tools for 200
- Help Compiler, WinHelp 257
- Help compiler, WinHelp
 - obtaining 58, 243
 - running automatically 248
- Help systems, merging 241
 - Eclipse Help 415
 - HTML Help 339
 - JavaHelp, Oracle Help 400
 - OmniHelp 366
 - WinHelp 249
- Help Viewer, Microsoft
 - index terms for 211
 - tools for converting CHM files 200
- Help Workshop
 - downloading 58, 243
 - for HTML Help 296
 - for WinHelp 54, 58, 243, 257
- Help, on-line 199
 - contents entries, configuring 209
 - contents levels, checking 209
 - context-sensitive, setting up 239
 - Eclipse Help
 - evaluating 202
 - generating 403
 - evaluating features of 200
 - HTML Help
 - evaluating 201
 - generating 295
 - HTML-based Help
 - contents levels, setting 210
 - contents list, maintaining 208
 - index entries, configuring 211
 - JavaHelp or Oracle Help
 - evaluating 202
 - generating 373
 - merging systems 241
 - Microsoft Help 2, tools for 200

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- OmniHelp
 - evaluating 202
 - generating 341
- Oracle Help for Java, evaluating 202
- related-topic links, providing 219
- topic levels, checking 203
- WinHelp
 - contents levels, setting 209
 - evaluating 200
 - generating 243
- HelpMerge**, HTML custom marker type 833
 - for HTML Help 339
 - for OmniHelp 368
- helpset file, JavaHelp, configuring 382
- helpsets, merging 400
- hexadecimal numbers
 - displaying 815
 - in results of expressions 811
- .hha file for HTML Help 330
 - location of 1024
- .hhc file
 - for HTML Help 321, 322
 - location of 1024
- .hhk file, location of 1024
- .hhp file for HTML Help 301, 304, 335
 - location of 1024
- HHReg, HTML Help tool 335
- .hht file, CSH IDs for HTML Help 331
 - location of 1024
- HHW, *see* **HTML Help Workshop**
- HIDC_ prefix for context-sensitive Help IDs 330, 365
- hidden text in Word
 - turning off for reviewers 144
 - used by **Mif2Go** 173
- hiding content in Word 173
- hiding white text for RTF output 173
- hierarchical links in HTML 627
- high ASCII characters
 - encoding for HTML 432
 - encoding for XML 460
 - mapping to HTML 653
 - mapping to RTF 172
 - replacing for W3C validation 454
- highlighting search terms in OmniHelp 364
- .hlp files for WinHelp 257
 - location of 1022
- home directory, Omni Systems, creating 54
- hotspots
 - See also* **image maps**
 - creating, for hypertext links 72
 - HTML Help, span of 226, 306
 - HTML, creating, for graphics 722
 - spanning entire paragraph 138
 - WinHelp
 - defining 274
 - for jumps and pop-ups 272
- hover text, providing, for HTML 448
- .hpj file for WinHelp 246
 - location of 1022
- .hs file for JavaHelp, location of 1024
- htm, DCL output type 1000
- .htm, default HTML file extension 427
- HTMConfig**, HTML custom marker type 833
- HTML
 - See also* **HTML code insertion**
 - content for database input 450
 - conversion files 1022, 1025
 - conversion template 426
 - converting to 423, 703
 - extracts
 - code insertion methods for 598
 - custom markers for 602
 - customizing 601
 - graphics in, referencing 607
 - replacing in parent file 603
 - thumbnails for reference to 604
 - titles, customizing 602
 - file extension, specifying 427
 - files, split and extract, referencing 600
 - generator, specifying 433
 - links, creating 609
 - lists, indenting 678
 - macros for
 - defining and invoking 787
 - including in a library 794
 - selectively enabling 791
 - using expressions in 811
 - using variables in 795
 - navigation macros 627
 - output
 - using FileIDs for 119
 - tables, *see* **tables: HTML**
 - tags, closing: suppressing 437

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- using XHTML tagging for 428
- HTML code insertion
 - in splits and extracts 598
 - keyword prefixes for splits and extracts, *listed* 599
 - keywords and locations, *listed* 599
 - keywords for splits and extracts, *listed* 600
 - methods for extracts, *listed* 598
 - pass-through, in FrameMaker 650
- HTML Help
 - See also* [HTML-based Help](#)
 - advantages and disadvantages of 201
 - ALink jumps, configuring 223
 - ALinks, target-and-jump 224
 - binary TOC
 - for browse buttons 303, 320
 - mid-topic links 323
 - no-link contents entries 322
 - browse buttons, enabling 303
 - .chm file, specifying 336
 - .chm, unblocking 296
 - compiling
 - and testing 333
 - for delivery 971
 - from **Mif2Go** 333
 - with HTML Help Workshop 334
 - contents, table of 319
 - customizing 324
 - files, generating 319
 - links to mid-file topics 323
 - merging entries 208
 - conversion files, location of 1024
 - file name restriction 52, 65
 - font resizing 303
 - framesets in 452
 - full-text search, providing 326
 - generating 295
 - contents and index files 319
 - href links to other .chm files 308
 - hypertext jumps to other .chm files 307
 - indents, eliminating 303
 - index entries
 - case sensitivity of, specifying 217
 - levels, combining 213
 - maximum length of 212
 - merging 208, 216
 - sort order, specifying 216
 - index files, generating 319
 - index, customizing 324
 - jumps to secondary windows 317
 - KLink jumps, configuring 223
 - links
 - specifying syntax of 308
 - to external files, configuring 308
 - map files, specifying 336
 - mapping FrameMaker files to CHM files 336
 - merging CHM files 339
 - parameters for ActiveX controls 317
 - pop-ups
 - creating with HTML Help 306
 - creating with KeyHelp 306
 - creating with WinHelp 307
 - in hypertext **Alert** markers 226, 306
 - project title, specifying 300
 - project, compiling 296
 - registering a CHM for network use 335
 - related topics, configuring 317
 - span of hotspots, determining 226, 306
 - starting topic, specifying 301
 - synchronizing TOC references 338
 - TOC, binary, compiling 305
 - uncompiled, configuring links for 308
 - viewer
 - for .chm files 296
 - using CSS with 303
- HTML Help Workshop 296
 - downloading 58
- HTML-based Help
 - See also:*
 - [Eclipse Help](#)
 - [HTML Help](#)
 - [JavaHelp](#)
 - [OmniHelp](#)
 - [Oracle Help for Java](#)
 - ALink jumps, configuring 223
 - ALinks, target-and-jump 224
 - checking automatic Help level assignments 586
 - contents levels, setting 210
 - index entries
 - case sensitivity, specifying 217
 - sort order, specifying 216
 - index link destination, specifying 215
 - KLink jumps, configuring 223
- HTMLComment**, HTML custom marker type 833
- HVIndex**, HTML custom marker type 211, 833
- hypergraphic, WinHelp graphic with hotspots 275
- HyperHelp, Bristol, *see* [Bristol HyperHelp](#)
- hyperlinks, *see* [hypertext](#), [links](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

hypertext

alert markers, *see* **alert** markers

commands, remapping as marker types 837

hotspots, creating 72

links

See also links, hypertext; links, HTML 622

active area of 138

broken, checking for 112

converting to HTML 619

for print RTF, converting 178

for print RTF, external 179

HTML, problem characters in 612

WinHelp, using for jumps and pop-ups 276

markers, how to insert 935

message commands

for HTML 181, 278, 625

hyphens, hard, in WinHelp 257

I

icons for drop-down links, configuring 231

id, HTML table attribute for WAI 759, 1013

purpose 763

via **CellID** marker 772

identifying

graphics files 133

Help elements, with character formats 933

Help files and titles, WinHelp 288

HTML files 120, 1027

links to other files, HTML 621

ideographic space in Japanese HTML Help 333

IDH_ prefix for context-sensitive Help IDs

for HTML Help 330

for OmniHelp 365

id/headers method, WAI

column and row identifiers, naming 782

columns, identifying 782

group identifiers, naming 779

identifying row and column groups 779

markup for table cells 777

markup for tables 763, 765

rows, identifying 783

span identifiers, naming 781

using span IDs 783

IDs

DITA element, specifying 495

DocBook element, specifying 571

file and object, *see* **FileIDs**; **ObjectID**

HTML table, *see* **TableID**

symbolic, for HTML Help CSH 327

symbolic, for OmniHelp CSH 364

.idx files for Oracle Help 389

<\$_if not>, control structure for macros 815

<\$_if>, control structure for macros 815

IGES, graphics export format 130, 884

Illustrator, Adobe, for converting graphics 130

image attributes

See also **graphics, attributes**; ** tag attributes**

DPI, including for DITA XML 518

omitting

for DITA XML 518

for DocBook XML 582

for generic XML 464

for HTML 720

specifying, for HTML 718

image maps, creating, for HTML 722

image, background, for HTML 725

ImageMagick, EPS graphics converter 876

images, *see* **graphics**

** tag attributes**

alignment 714

class for anchor paragraphs 693

for *Made with Mif2Go* graphic 453

specifying 718

via markers 756, 835

src, specifying 705, 877

for JavaHelp 381

** tag class for anchor paragraphs** 463

importing

DITA output into FrameMaker 8 481, 519, 521

formats from a conversion template 863, 936

INCLUDEPICTURE field for Word external graphics

188, 873, 877, 904

indenting

graphics, HTML 716

list items, HTML 678

tables, HTML 747

index

See also **index entries**

activating hypertext links in 125

converting from FrameMaker IX 81, 124

crash 125, 1036

for HTML 442

for Word 195

files, generating

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- for HTML Help 319
 - for JavaHelp 385
 - for OmniHelp 356
 - for Word, in Word 195
 - link destinations for HTML-based Help, specifying 215
 - links, activating 125
 - for Word 182
 - markers
 - converting to Word {xe} fields 195
 - mapping, for Help systems 140
 - mapping, for HTML 442
 - treating content as text 844
 - page numbers
 - applying a character format to 125
 - in Word, accuracy of 182
 - replacing for HTML 442
 - properties, configuring for Eclipse Help 414
 - ranges
 - expanded for Help systems 212
 - omitting for HTML-based Help 212
 - sort order, specifying
 - for HTML Help, OmniHelp 216
 - Japanese, for Help systems 218
 - terms, *see* [index entries](#)
 - Word
 - generating in Word 195
- index entries
- See also* [index](#)
 - for DITA XML, sorting 485
 - for Eclipse Help
 - configuring 414
 - special characters in 412
 - for Help systems, configuring 211
 - for HTML Help, maximum length of 212
 - for HTML-based Help
 - case sensitivity of 217
 - configuring 211
 - level separators for 213
 - merging from multiple files 208
 - range, automatic 212
 - range, omitting 212
 - sort order of, specifying 216
 - sort strings in, using 218
 - for JavaHelp
 - configuring 386
 - merging from multiple files 208
 - for Microsoft Help Viewer, preparing 211
 - for OmniHelp, configuring 359
 - for print RTF, ensuring targets for 183
 - for WinHelp, level separators for 287
 - for XML, from index markers 468, 844
 - <\$nopage>, *see* <\$nopage> index entries 215
- IndexRef, for *See* and *See also* entries 125
- for HTML 444
 - for Word 184
- index.xml, Eclipse Help index file 414
- indirect references, *see* [pointers](#)
- indirection, using in macro expressions 819
- .inf files
 - creating 1001
 - document information 1017
- inf, DCL output type 1000
- infopops for Eclipse Help, configuring 417
- .ini file, *see* [configuration file](#)
- in-line graphics
 - in Word, preserving borders of 900
- inline text, content-model element type 912
- insets
 - FrameMaker, *see* [text insets](#)
 - HTML, importing files as 446
- installing **Mif2Go**
 - for the first time 56
 - updates 61
- instructions, adding with markers 935
- interfile links
 - in HTML, to renamed files 622
- interfile links in HTML
 - See also* [reference files for HTML](#)
 - to files in other projects 623
- invisible paragraphs, eliminating, HTML 652
- IX, *see* [index](#)

J

- Japanese
 - combined fonts option 80
 - for HTML Help output, specifying 332
 - for HTML Help, compiling 335
 - for RTF output, specifying 147
 - ICU DLLs for HTML output 54
 - ICU DLLs, obtaining 300
 - index sort order for on-line Help 218
 - Mif2Go** support for 53
- JAR file, creating 390

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Java Runtime Environment, for JavaHelp 54

Java Virtual Machine, for JavaHelp 373

JavaHelp

See also [HTML-based Help](#)

advantages and disadvantages 202

ALink jumps, configuring 223

compiling, with Helen 384

contents and index files

creating 385

locating 387

conversion files, location of 1024

conversion, setting up directories for 374

excluding face attribute of font tags 665

full-text search 387

generating 373

glossary, converting to 392

helpset file, configuring 382

images, mapping to files 394

index entries

case sensitivity of, specifying 217

configuring 386

index link destination, specifying 215

JAR file, creating 390

JHIndexer command 388

map file, specifying location of 381

version 2.0, downloading 58

windows, defining 393

JavaScript

for expandable sections 234

including in HTML output 52, 424

inserting, for HTML attributes 436

using macro variables in 600

JH2Pop*, custom markers for JavaHelp 2 pop-up window properties 833

JH2Sec*, custom markers for JavaHelp 2 secondary window properties 833

JHIndexer command for JavaHelp 388

.jhm file, JavaHelp map file 401

location of 1025

JPEG graphics export format 130, 884

for Web use 871

.jpg files

exporting 881

for DITA XML, location of 1023

for DocBook XML, location of 1023

for HTML, location of 1023

JRE, Java Runtime Environment 54

jumps

ALink

configuring for Help systems 223

macros for HTML Help 312

with keywords for HTML Help 312

and pop-ups, WinHelp

coding character formats for 281

creating 272

local, coding 283

using hypertext links for 276

destinations of, WinHelp

cross-reference, specifying 260

external, coding 284

KLink, configuring for Help systems 223

related-topic, adding for Help systems 222

to other Help files, HTML Help 307

to secondary windows

in Help systems 224

in HTML Help 317

in OmniHelp 360

Oracle Help 399

K

key names in configuration settings 104

Key Tools, obtaining 306

KeyHelp, DLL for HTML Help pop-ups 306

KeyrefBranch, PI for keyref to named map branch 833

keyword links, *see* [KLinks](#)

keywords, configuration

DITA content model, *listed* 1043

DITA, *listed* 1039

HTML, *listed* 1059

RTF, *listed* 1047

KLinks

access to merged topics 220

HTML-based Help, configuring 223

jump destinations of, specifying 224

maintenance issues 221

OmniHelp, support for 359

understanding 221

WinHelp, limitations of 285

Korean

for HTML Help output, specifying 332

for RTF output, specifying 147

Mif2Go support for 53

ABCDEFGHIJKLMNOPQRSTUVWXYZ

L

- label attribute for Eclipse Help index entries [414](#)
- label, Eclipse Help TOC, for book level [413](#)
- landscape
 - pages, converting to RTF [157](#)
 - tables, converting to RTF [186](#)
- language, output, specifying
 - for HTML Help [331](#)
 - for <html> tag [430](#)
 - for print RTF [147](#)
- languages supported [53](#)
- last, macro string operator [817](#)
- \$_lastfile, macro variable [601](#), [801](#)
- <\$_lastlocaltoc>, predefined macro for HTML [634](#), [792](#)
- layout options, RTF, specifying [151](#)
- leading, *see* [line spacing, adjusting](#)
- legacy content, migrating
 - to DITA [474](#)
 - to DocBook [558](#)
- length, macro string operator [817](#)
- levels, macro nesting [791](#)
- Leximation DITA-FMx plug-in [480](#), [481](#), [519](#), [521](#), [530](#), [541](#), [543](#)
- libraries, run-time
 - downloading [62](#)
 - listed* [1017](#)
- library, macro, creating and naming [794](#)
- license for **Mif2Go**, purchasing [1029](#)
- line breaks
 - in DITA <codeblock> elements, preserving [489](#)
 - in DITA, inserting via processing instructions [499](#)
 - in DocBook <programlisting> elements, preserving [566](#)
 - in HTML tables, forcing [744](#)
 - in HTML, suppressing [437](#)
 - in HTML/XHTML <pre> text, eliminating [670](#)
 - in macros, including or excluding [789](#)
 - in XML, suppressing [437](#), [461](#)
- line spacing, adjusting
 - for HTML list items [678](#)
 - for RTF [170](#)
 - in CSS [700](#)
- line wraps in <pre> text, eliminating [670](#)
- lines, dashed, in WMF graphics [870](#)
- Link***, custom markers for HTML link attributes [833](#)
- LinkClass** marker [610](#)
 - effect of [833](#)
 - for WAI [759](#), [1016](#)
- links
 - See also* [cross references](#); [hypertext](#), [links](#)
 - See also* [links, HTML](#); [hypertext](#), [links](#); [ALinks](#); [KLinks](#)
 - adding with markers [935](#)
 - broken, checking for [112](#), [1033](#)
 - hypertext
 - converting to HTML [619](#)
 - converting to RTF for Word [178](#)
 - determining active area of [138](#)
 - related-topic
 - ALinks and KLinks [219](#)
 - for on-line Help [219](#)
 - XML
 - anchors for, managing [467](#)
 - configuring [467](#)
- links, HTML
 - creating [609](#)
 - CSS class, assigning
 - via format [611](#)
 - via marker [610](#)
 - drop-down, configuring [230](#)
 - buttons [232](#)
 - icons [231](#)
 - text [232](#)
 - type, specifying [230](#)
 - drop-down, delimiting
 - with formats [228](#)
 - with markers [229](#)
 - for breadcrumb trails [627](#)
 - forcing to lowercase [613](#)
 - from cross references [617](#)
 - hierarchical [627](#)
 - keeping ObjectID [620](#)
 - mid-topic, from TOC [210](#)
 - navigation
 - behavior of [636](#)
 - creating [627](#)
 - suppressing in <\$nopage> index entries [444](#)
 - to footnotes, eliminating [672](#)
 - to other documents [1028](#)
 - to other files, identifying [621](#)
- \$_linksrc, macro variable [612](#), [619](#), [758](#), [801](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

LinkTitle, marker for WAI attribute 759, 1016

list formats

converting

to DITA XML 502, 505

to DocBook XML 574

to HTML 674

dictionary, converting

to HTML 677

to HTML tables 824

indenting, for HTML 678

nested, converting

to DITA XML 505

to HTML 676

list of figures or tables, converting

prevention of 125

to HTML 444

to HTML Help 325

to Word 181

list variables

creating with configuration sections 807

for macros 806

initializing 807

instead of conditional expressions 809

processing with macros 807

processing with pointers 808

using in expressions 818

literals, character

assigning to macro variables 798

for macro variables, *listed* 798

local contents for HTML 631

local jumps and pop-ups, WinHelp, coding 283

locale

for index sort order 218

for RTF output 147

identifier for HTML Help 331

specifying for HTML Help 331

<\$_localtoc>, predefined macro for HTML 634, 792

LocalTOCTitle, custom marker for HTML local TOCs 634, 833

\$_loctocfile, macro variable 633, 801

\$_loctoctype, macro variable 633, 801

log file

editor for displaying when errors 115

for conversion events 115

mif2go.log, **runfm** results 988

-log, **runfm** option 982, 988

logging

automation commands 956

conversion events 115

link errors 112

runfm commands and results 988

logical operators for macro expressions, *listed* 812

longdesc, HTML image attribute for WAI 757, 1015

loops, nesting, in macros 817

lower, macro string operator 818

.lst file

for DITA XML, location of 1023

for DocBook XML, location of 1024

for Eclipse Help, location of 1025

for HTML Help, location of 1024

for HTML-based Help, maintaining 208

for HTML/XHTML/XML, location of 1023

for JavaHelp, location of 1025

for OmniHelp, location of 1024

M

m2gframe.dll, **Mif2Go** plug-in interface 1017, 1034

m2g_log.txt, conversion event log file
default location of 1019

m2hmacro.ini, sample macros for HTML 1017

m2rbook.dll, **Mif2Go** plug-in interface 1017

macro

See also macros, **Mif2Go**

configuration file, editing 861

expressions, results of

displaying in output 813

interpreting 811

files, individual 793

macro libraries, organization of 851

macro parameter, passing 820

macro variables

See also macros, **Mif2Go**

assigning paragraph content to 803

assigning values to 797

assignment values of, displaying 798

from FrameMaker user variables 802

in HTML navigation macros 638

incrementing and decrementing 799

list type 806

See also list variables

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- using in expressions 818
- nesting 798
- predefined
 - for HTML extract replacement, *listed* 603
 - for HTML splits and extracts, *listed* 601
 - for system commands 939
 - listed* 800
 - uses for 800
- referenced in WAI attributes 755
- syntax of 796
- undefined, debugging 820
- valid contexts for 821
- \$\$_macroparam, macro variable 801
- macros, **Mif2Go**
 - See also* [macro](#); [macro variables](#)
 - backslash escape character in 789
 - conditional expressions in 815
 - control-structure elements, *listed* 815
 - debugging 820
 - defining 787
 - expression results 811
 - expressions 811, 820
 - for HTML
 - framesets, using to create 450
 - inserting, for split and extract files 598
 - insertion methods for extracts 598
 - JavaHelp secondary windows and pop-ups 393
 - navigation, inserting predefined 635, 821
 - navigation, redefining 639
 - referenced in WAI attributes 755
 - table, specifying 748
 - using for attribute text 619
 - using for link properties 612
 - using to specify WAI attributes 761
 - for system commands 940
 - line breaks in 789
 - nesting 791
 - nesting limit 792
 - operands 811
 - operators, *listed* 812
 - predefined, HTML *listed* 792
 - specifying where to invoke 821
 - ternary operators '?' and ':' 816
 - trailing spaces at end of 789
- macros, WinHelp, invoking 284
- <\$_madewith>, predefined macro for HTML 792
- madewithm2g.jpg, “Made with **Mif2Go**” icon 452
- Maker Interchange Format, *see* [MIF](#); [.mif](#)
- manifest file, Eclipse Help
 - MANIFEST.MF, configuring 408
 - plugin.xml, configuring 409
- MANIFEST.MF, Eclipse Help manifest file 407
- map files for context-sensitive Help 240
 - HTML Help, specifying 336
 - JavaHelp, specifying location of 381
- [MAP] section of HTML-based Help file 329
- maps
 - ditamaps, configuring 539
- margins
 - specifying, for HTML Help pop-ups 306
 - unusually large, in RTF 151
- marker types
 - See also* [markers](#)
 - assigning properties to 838
 - cloning 139
 - configuration, defining 921
 - creating and cloning 139, 836
 - effects of properties, *listed* 839
 - hypertext, how to insert 935
 - mapping 139
 - naming 834
 - predefined
 - for all output types, *listed* 832
 - for DITA maps, *listed* 556
 - for DITA XML, *listed* 536
 - for DocBook XML, *listed* 583
 - for HTML extracts, *listed* 602
 - redefining 840
 - suppressing 841
- markers
 - adding links and instructions with 935
 - ALink**, for Help systems 140
 - attribute, for HTML or XML
 - See also* [attribute markers for HTML or XML](#)
 - applying 835
 - for images 718
 - for links 612
 - for tables 737
 - including in macro code 845
 - listed* 835
 - understanding 834
 - configuration, to override settings 921
 - cross-reference, eliminating Word-generated 114
 - custom
 - adding in FrameMaker 832
 - for HTML extracts 602

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- custom, WAI
 - advantages of 760
 - for image attributes 1014
 - for link attributes 1016
 - for table attributes 772, 1013
 - using to specify attributes 756
 - hypertext
 - commands, remapping 837
 - how to insert 935
 - hypertext **alert**
 - and **alerttitle**, for WinHelp pop-ups 277
 - for HTML Help pop-ups 226, 306
 - for HTML split points 587
 - for splitting HTML files 587
 - identifying, with \$\$_objectid 847
 - index, *see* [index](#), [markers](#)
 - mapping
 - for Help systems 140
 - for HTML 442
 - repurposing 139
 - text, cross-reference, truncating 261
 - Type 11, for WinHelp mid-topic jumps 277
- marklib.fm, WAI marker library 1013
- master pages
 - content omitted for HTML 645
 - cross references, for print RTF 181
 - different size and orientation 157
 - including graphics on
 - for HTML 885
 - for Word 154
 - layout restrictions for Word 151
 - sidehead width, adjusting for Word 153
 - substituting via conversion template 68, 151
- MathFullForm for FrameMaker equations, in DCL output 136
- MathFullForm objects for equations
 - included in DCL 136
 - including in DITA `alt` tags 518
- memory deallocation 1035
- merging
 - Eclipse Help projects 415
 - Help systems 241
 - HTML Help .chm files 336
 - JavaHelp or Oracle Help systems 400
 - OmniHelp projects 366
- message** **openfile** hypertext links
 - for HTML 625
 - for WinHelp 278
 - for Word 181
- how to insert 935
- message** **URL** hypertext links
 - for HTML 625
 - for WinHelp 278, 284
 - for Word 181
 - how to insert 935
- Meta***, custom markers for `<meta>` tag content 833
- `<meta>` tag content, supplying 434
 - for split or extract files 598
- metafiles
 - embedding bitmap graphics in 886
 - embedding equations in 137
 - internal vs. external, specifying 873
 - naming, for WinHelp 134
- metrics, font, specifying for RTF 165
- Microsoft
 - Help Workshop, *see* [Help Workshop](#)
 - HTML Help Workshop, *see* [HTML Help Workshop](#); [Help Workshop](#)
 - HTML Help, *see* [HTML Help](#)
 - Vista, no support for WinHelp 200
 - Word, *see* [Word](#)
- Microsoft Help Viewer
 - index terms for 211
 - tools for converting CHM files 200
- mid-topic entry points
 - for Eclipse Help context anchors 418
 - for HTML Help CSH links 327, 330
 - for index links, not recognized by RoboHelp 216
 - for TOC links, in HTML Help 323
 - incompatible with HTML Help binary TOC 303, 323
- mid-topic headings, moving link anchors above 615
- mid-topic links
 - from OmniHelp TOC, avoiding 354
 - in Eclipse Help TOC, enabling 413
 - in Help systems, effects of 200
- MIF files
 - automatically creating/deleting, via plug-in 992
 - deleting after conversion 85
 - existing, using 85, 111
 - file extension, specifying 1008
 - generating with **Mif2Go** 1006
 - managing 111
 - old, deleting before converting 960
 - to clean corrupt .fm files 1005, 1032
 - version 8, from FrameMaker version 8 1008

ABCDEFGHIJKLMNOPQRSTUVWXYZ

.mif, MIF (Maker Interchange Format) files 62

MIF, saving as, via **Mif2Go** 1006

Mif2Go

command-line applications

DCL 995

runfm 979

getting started with 51

installing 56

plug-in, using 77

running

via DCL 995

via FrameMaker plug-in 77

via **runfm** 979

stopping 63

uninstalling 64

updating 61

mif2go.ini, configuration file

combined 623

for FileIDs 120

providing for batch conversions 997

renaming or relocating 1027

mif2go.log, **runfm** results 982, 988

mif2go.prj, fallback project file 1026

for **runfm** 983

modular Help systems 241

modules, DCL conversion, writing 1003

moving files

conversion 63, 1025

FileID 1027

Mif2Go project 1026

MS HTML Help, *see* [HTML Help](#)

msvcrt40.dll, Microsoft Windows C++ run-time library 1017

N

named destinations from Word cross references 114

names

See also [naming](#)

of CSS classes, case sensitivity 692

of files and paths, restrictions on 52, 65

of files, in double quotes 999, 1001

of FrameMaker formats, restrictions on 66

namespace, HTML, specifying 430

naming

See also [names](#)

files

and paths 51, 65

graphics 134, 1010

helpset, JavaHelp 382

HTML FileID and project 1026

HTML split and extract 946

WinHelp 288

WinHelp topic 260

marker types 834

projects

Eclipse Help 407

Mif2Go conversion 78

OmniHelp 347

WinHelp primary window 291

navigation

buttons

for HTML 638

for HTML Help 303

for OmniHelp 353

links for HTML, creating 635

macros for HTML

button definitions, *listed* 642

buttons for 638

default definitions of 637

redefining 639

scope of 642

text-link definitions, *listed* 642

where to invoke 642

titles

for DITA topics, alternate 526

Ndoc, for Microsoft Help Viewer 200

nested FrameMaker books 53

nested lists

converting to DITA XML 505

converting to HTML 676

nesting

DITA elements 477, 501, 505

DITA maps 540

DITA topics 521

DocBook elements 573

macro loops and conditionals, forbidden 815, 816

macro variables 798

macros 791

network drive

for shared configurations 855

not a good place for %OMSYSHOME% 54

network drives

why not to use 1032

ABCDEFGHIJKLMNOPQRSTUVWXYZ

network file system
 operating **runfm** across 992
 problems accessing files 74, 1030

network file system, using HTML Help across 335

.new, extension for changed configuration files 92

newlink markers
 for DITA
 cross references 477
 ID attributes 495
 for DocBook 571
 for HTML Help CSH 327
 for HTML hypertext links 619
 for OmniHelp CSH 364
 for WinHelp
 in marker lists 267
 pop-ups 265, 275, 279
 using Type 11 markers for 277
 how to insert 935

<\$_next>, HTML navigation macro 635, 792

\$\$_nextfile, macro variable 601, 801

\$\$_nexttitle, macro variable 601, 801

<\$nopage> index entries
 for HTML 444
 for HTML-based Help 215
 for OmniHelp 359
 for Word 184
 for Word-generated index, discarding 196

no-scroll region for WinHelp topic titles 271

numbered lists, converting
 to HTML 676
 to RTF 158

numeric entity references
 for HTML 432
 for XML 460
 in Eclipse Help contents or index 412

numeric IDs for context-sensitive Help 240

O

ObjectID
 determining for HTML split files 594
 duplicate, removing 119
 FrameMaker 118
 links, keeping, HTML 620
 specifying for print RTF 175, 249

\$\$_objectid, macro variable 801
 in file names 951

to identify markers 847

OLE objects
 exporting all WMF images 882
 extracting images with FrameMaker filters 882
 extracting WMF previews 881
 retrieving for use in original application 882

Omni Systems
 environment variable %OMSYSHOME%, creating 55
 home directory, creating 54

OmniHelp
 See also **HTML-based Help**
 advantages and disadvantages of 202
 ALink jumps, configuring 223
 ALink keywords, displaying 359
 ALinks, target-and-jump 224
 buttons, excluding or displaying 353, 359
 contents
 expanding and collapsing 357
 including 356
 context-sensitive Help, setting up 364
 conversion files, location of 1024
 cookies, persistence of 371
 CSS usage, specifying 350
 data and control files, *listed*
 generated by **Mif2Go** 345
 supplied in ohview.zip 344
 file name restriction 66
 files, obtaining 342
 frameset and frame dimensions, specifying 352
 full-text search
 configuring 361
 including 356
 terms, highlighting 364
 index entries
 case sensitivity of, specifying 217
 expanding and collapsing 357
 levels, combining 213
 See and See also entries 359
 sort order, specifying 216
 index link destination, specifying 215
 index, including 356
 interface, localizing 202, 354
 KLink jumps, configuring 223
 launching 370
 memory requirements 348
 navigation aids, modifying 353
 navigation panel, modifying 355
 pop-up windows, specifying 360
 prev/next buttons, including 354

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- projects
 - merging 366
 - naming 347
 - setting up 345
 - titles of 348
 - related topics
 - including 356
 - links, providing 359
 - search terms, highlighting 364
 - secondary windows, jumping to 360
 - settings, making persistent 354
 - starting topic, specifying 348
 - template, modifying 356
 - %OMSYSHOME% environment variable 55
 - on-line Help, *see* [Help, on-line](#)
 - openlink** markers
 - converting to HTML links 609, 619
 - how to insert 935
 - specifying a destination for HTML 619
 - using for WinHelp jumps and pop-ups 276
 - OpenOffice, producing RTF for 197
 - operating settings, specifying 109
 - operators for macro expressions, *listed* 812
 - Oracle Help for Java
 - See also* [HTML-based Help](#)
 - advantages and disadvantages of 202
 - ALink jumps, configuring 223
 - ALinks, target-and-jump 224
 - content and index, creating 387
 - Developer's Kit 54
 - downloading 58
 - full-text search 387
 - index entries
 - case sensitivity of 217
 - configuring 386
 - index link destination, specifying 215
 - JAR file, creating 390
 - obtaining information about 373
 - windows, defining 393
 - order of configuration-file sections and settings 103
 - output
 - directory, specifying for project via plug-in 78
 - file paths and names, specifying 1002
 - format, specifying 78
 - HTML, using FileIDs for 119
 - type, specifying
 - for print RTF 147
 - for WinHelp 248
 - underline, replacing with a tag in HTML/XML 669
 - overrides
 - See also* [overriding](#)
 - configuration
 - See also* [configuration settings, overriding](#)
 - for HTML table and graphics groups 930
 - persistent vs. temporary 921
 - format
 - allowing or eliminating for HTML 657
 - retaining in HTML for structured documents 135
 - suppressed for DITA XML 494
 - overriding
 - configuration settings
 - fixed-key 924
 - in macros 921
 - variable-key 925
 - with command-line options 998
 - with configuration markers 921
 - with text, for HTML 931
 - cross-reference properties 928
 - format properties 926
 - HTML graphics properties 929
 - HTML table
 - [Attributes] values 750
 - column and row groups 733
 - default heading/footer counts 735
 - default WAI cell settings 784
 - display attributes 736
 - properties 928
 - WAI markup method 765
 - paragraph properties for HTML 653
 - path to graphics for HTML 888, 929
 - split points in HTML 588
 - overview topic in WinHelp 290
- ## P
- </p> tags, suppressing in HTML 437
 - page
 - breaks
 - See also* [breaks, page](#)
 - automatic, keeping the same in Word 152
 - handling for print RTF 152
 - handling for WinHelp 249
 - inserting in DITA via PIs 499
 - using for HTML split points 587
 - numbers
 - eliminating from cross references 68

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- in cross references, for print RTF 180
 - index, applying a character format to 125
 - index, replacing for HTML 442
 - retained, in XML output 468
 - suppressing, for HTML TOC 445
- size and orientation, converting to RTF 157
- titles, for HTML files 594
 - assigned with markers 597
 - based on file names 596
 - based on paragraph formats 595
 - computed 597
 - default 597
 - precedence 595
 - prefixes and suffixes 596
 - related to StartingSplit 588
- pagination, maintaining in Word 152
- Paint Shop Pro, for converting graphics 130
- paragraph
 - See also* paragraphs
 - anchors, converting to RTF 171
 - attributes, suppressing, for HTML 650
 - autonumbers, eliminating, for HTML 465, 649
 - formats
 - See also* formats, paragraph
 - deleting, for WinHelp 253
 - eliminating tags for HTML 650
 - for graphics, specifying 191, 264
 - for splitting HTML files 586
 - mapping to DITA elements 487
 - mapping to DocBook elements 565
 - mapping to RTF styles 158
 - merging, RTF 159
 - properties of, overriding 926
 - replacing with code, for WinHelp 257, 822
 - replacing with code, for Word 174, 822
 - replacing with comments for HTML 650
 - run-in, converting to HTML 648
 - script, designating, HTML 650
 - suppressing, for WinHelp 253
 - properties
 - changing for individual paragraphs 926
 - overriding, HTML 653
 - stripping, HTML 650
 - spacing, adjusting for RTF 170
- paragraphs
 - See also* paragraph
 - empty
 - keeping or removing in Word 171
 - omitting tags for in HTML 652
 - providing content for in HTML 651
 - replacing with RTF code
 - for WinHelp 257, 822
 - for Word 174, 822
 - unwanted, eliminating for HTML 652
 - ParaID, FrameMaker ObjectID for paragraphs 118, 594
 - parameter entities, equivalent to element sets 910
 - parameter for **Mif2Go** macro 820
 - parameter lists for DITA output 505
 - <\$paranum> and <\$paranumonly>, replacing for Word 181
 - \$_paratag, macro variable 801
 - \$_parauid, macro variable 615, 801
 - pass-through code, HTML 650
 - path
 - See also* file, paths in configuration settings
 - current, macro variable for 601
 - default, for Word documents 181
 - names
 - restrictions on 52, 65
 - spaces in, avoiding 52, 65, 1032
 - omitting from links, for OmniHelp 349
 - overriding, for HTML graphics 888, 929
 - relative vs. absolute
 - in configuration settings 105
 - in graphics references 705
 - retaining in interfile links for HTML 622
 - specifying, for HTML graphics 705
 - to assembly directory 961
 - to configuration template 851
 - to conversion template, specifying 864, 866
 - to CSS directory, for copying CSS files 969
 - to graphics files
 - on UNIX server 705
 - removing, for HTML 704, 887
 - to project directory, macro variable for 800
 - to shipping directory 975
 - .pct files, exporting 881
 - .pcx files, exporting 881
 - PDF output, generating via **runfm** 985
 - pdf**, **runfm** option 981, 985
 - pdfsave**, **runfm** option 981
 - pen style patterns, FrameMaker, mapping 900
 - pernicious mixed content
 - constraining in DITA output 477

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- persistent configuration overrides 921
- persistent settings in OmniHelp 354
- PICT, graphics export format 130, 884
- pictures, *see* [graphics](#)
- pkzip.exe
 - for archiving deliverables 973
 - for packaging Eclipse Help topic files 419
- placement of, for HTML/XML 671
- platform differences, accommodating, WinHelp 247
- plug-in manifest file `plugin.xml`, Eclipse Help, configuring 409
- plug-in, FrameMaker
 - DITA-FMx, *see* [DITA-FMx plug-in, Leximation IndexRef 125, 184, 444](#)
 - Mif2Go** 77
 - SetPrint 987
- `plugin.xml`, Eclipse Help manifest file 409
- `.png` files, exporting 881
- PNG, graphics export format 130, 884
- point size, matching for graphics 901
- pointers
 - to process lists 808
- pop-ups
 - See also* [windows, pop-up](#)
 - browser, suppressing, effect on OmniHelp 372
 - HTML Help 305
 - creating with HTML Help 306
 - creating with KeyHelp 306
 - creating with WinHelp 307
 - HTML, require JavaScript 616
 - JavaHelp, using macros for 393
 - OmniHelp, specifying 360
 - WinHelp
 - alert, creating 277
 - creating 272
 - from table cells 263
 - hotspots for 274
 - local, coding 283
 - using hyperlinks for 276
- postprocessing
 - activating and logging 956
 - automated 955
 - choosing options for, on Export 88
 - files copied, *listed* 964
 - graphics files copied, *listed* 966
 - separately from converting 976
 - understanding 955
- `pprtf.exe`, RTF pretty printer 1017
- `<pre>`, HTML paragraph tag 647
- precedence
 - of configuration settings 852, 862, 919
 - listed* 920
 - of DITA topic type assignments 524
 - of extract code insertion methods 598
 - of extract property assignments 602
 - of HTML page title assignments 595
 - of macro definitions 792, 795
 - of macro variable definitions 796
 - of shading colors in HTML tables 745
 - of table property assignments 728
- predefined
 - macro control-structure elements, *listed* 815
 - macro variables
 - all, *listed* 800
 - for HTML splits and extracts, *listed* 601
 - in system commands 939
 - using 800
 - macros, *listed* 792
 - marker types, *listed* 832
- preformatted text
 - assigning HTML `<pre>` tags 646
 - content-model element type 912
 - empty tags preserved in 652
 - HTML/XHTML, configuring 670
 - in table cells
 - for DITA 513
 - for DocBook 564
 - for HTML/XML 744
 - preserving whitespace in for DITA 499
- `<$_prev>`, HTML navigation macro 635, 792
- `$$_prevfile`, macro variable 601, 801
- `$$_prevsfile`, macro variable (*deprecated*) 601
- `$$_prevtitle`, macro variable 601, 801
- primary window, naming in WinHelp 291
- print file, naming, for `runfm` 984
- print options, FrameMaker, set by `runfm` 986
- `-print`, `runfm` option 981, 984
- printable set, characters in 658
- printer
 - Adobe PDF, configuring for `runfm` 985
 - default, specifying via SetPrint 987
 - for `runfm`, specifying 987
- `-printer`, `runfm` option 981, 987
- printing via `runfm` 984

ABCDEFGHIJKLMNOPQRSTUVWXYZ

.prj file
 See also [project file](#)
 location of [1019](#)
 Mif2Go conversion projects [78](#)
 role in converting individual chapters [937](#)

\$\$_prjname, macro variable [801](#)

\$\$_prjpath, system-command variable [801](#), [939](#)

processing instructions in DITA, for line and page breaks [499](#)

-progid
 FrameMaker RPC option [980](#)
 runfm option
 for **Mif2Go** [981](#), [982](#)
 for other plug-ins [993](#)
 for remote operation [992](#)

project directory, establishing [74](#)

project file
 HTML Help [301](#), [304](#), [335](#)
 location of [1024](#)
 JavaHelp helpset [382](#)
 Mif2Go
 See also [.prj file](#)
 accessed by **runfm** [983](#)
 location of [62](#), [1019](#)
 WinHelp [246](#)
 location of [1022](#)
 naming [248](#)

project, **Mif2Go**
 name, specifying for **runfm** [983](#)
 naming [78](#)
 setting up [78](#)
 for multiple books [75](#)

-project, **runfm** option
 for **Mif2Go** [981](#), [983](#)
 for other plug-ins [993](#)

properties, document, importing [865](#)

public and system IDs, overriding [915](#)

PUBLIC declaration for HTML/XML [429](#)

punctuation
 in ALink keywords, avoiding
 for HTML Help [309](#)
 for OmniHelp [359](#)
 for Oracle Help [399](#)
 in CSH **newlink** markers
 allowing [241](#)
 for JavaHelp, Oracle Help [401](#)
 in file and directory names, avoiding [51](#), [65](#)
 in format names, avoiding [66](#)

in HTML file names [948](#)
 in index entries
 detecting for XML tags [470](#)
 ignoring for sort order [217](#)
 in link keywords, disallowed [219](#)

Q

quotes
 around configuration-assignment values [922](#)
 around macro names in overrides [923](#)
 baseline, converting to straight, in macros [790](#)
 smart, converting
 for WinHelp [256](#)
 in macros [790](#)
 style, specifying for print RTF [172](#)

R

raster graphics, *see* [bitmaps](#); [graphics](#), [bitmap](#)

raw code, HTML, including in FrameMaker [650](#)

redirect pages for OmniHelp CSH [365](#)

.ref files, interfile links
 location of [1019](#), [1023](#)
 understanding and using [1027](#)
 when not to delete [959](#)

reference files for HTML [1027](#)
 See also [interfile links in HTML](#)
 deleting between conversions [427](#), [958](#), [1026](#)

reference frames
 converting to RTF [162](#)
 converting to WinHelp [253](#)
 removing for HTML [651](#)

reference page graphics
 including or excluding
 for HTML [885](#)
 for RTF [162](#)
 for WinHelp [253](#)
 written only if used [131](#)

references, updating before converting [126](#)

related topic
 keywords
 adding with format properties [222](#)
 adding with markers [140](#), [221](#)
 links
 ALinks and KLinks [219](#)
 for Help systems [219](#)

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- for HTML Help 309
 - for OmniHelp 356, 359
 - for Oracle Help 400
 - for WinHelp 285
 - in DITA maps 546
- relational operators for macro expressions, *listed* 812
- relationship tables, DITA 546
- relative vs. absolute paths
 - in configuration settings 105
 - in graphics references 705
- remote operation of **Mif2Go** via **runfm** 992
 - enabling DCOM for 992
- remote**, **runfm** option 981, 983, 992
- renaming files via system commands 937
- repeat loops in macros 816
- <\$repeat>, control structure for macros 815, 816
- replace with, macro string operator 818
- requirements, system 53
- rescaling bitmaps
 - see also* [scaling](#)
 - for RTF 898
 - for screenshots 872
 - via FrameMaker export filters 884
- resource.h, Help map file 240
- returns, hard, *see* [hard returns](#); [line breaks](#)
- returns, soft, *see* [line breaks](#); [soft returns](#)
- reverse**, **runfm** option 981
- review process using Word 144, 943
- revision system, checking files in and out of 979
- revision tracking in Word 144, 194, 944
- .rf files, exporting 881
- RGB colors
 - converted from CMYK
 - for HTML 438
 - for WinHelp 258
 - for Word 172
 - problems with 441
 - Web-safe 440, 746
 - listed* 440
- RoboHelp, for generating WebHelp 200, 201, 216
- root element
 - for content model 909
 - XML, specifying 458
- rotated table cells
 - in HTML 426
 - in Word 142, 185
- rotated text in callouts 191
- round-trip DITA output 480, 481
- row spans, identifying 781
- Row***, custom markers for HTML or XML table row attributes 833
- RowClass**, custom marker for CSS 738
- RowGroup cells 785
 - and ColGroup cells, using 784
 - with id/headers method 778
 - with scope attributes 776
 - defined* 767
- rowspan, HTML table attribute 760
- RTF
 - configuration file
 - location of 1022
 - conversion files, location of 1022
 - converting to 141
 - formats, mapping 158
 - raw code, replacing content
 - in WinHelp 257, 822
 - in Word 174, 822
- RTF code, including for Word 194
- rtf, DCL output type 1000
- RTFConfig**, RTF custom marker type 833
- runfm**
 - advantages of, for command-line use 992
 - command-line syntax 980
 - compared with DCL 991
 - console messages, reviewing 988
 - operating across a network 992
 - options and arguments, *listed* 981
 - using 979
 - for a series of conversions 990
 - for a single conversion 989
 - to run **Mif2Go** 982
 - to run other plug-ins 993
- runfm.exe, command-line application 1017
- run-in formats, converting
 - to HTML 648, 825
 - to print RTF 160
 - to WinHelp 252
- Running H/F variables, converting to RTF 156
- running headers and footers, *see* [headers](#)
- run-time libraries, **Mif2Go**, *listed* 1017

ABCDEFGHIJKLMNOPQRSTUVWXYZ

run-time values, supplying 941

Russian

for HTML Help output, specifying 332

for RTF output, specifying 147

Mif2Go support for 53

S

saving a project from within FrameMaker 79

Saxon, for DITA XML 475

SAppLocale, HTML Help compiler for other locales 335

scaling

See also [rescaling](#)

bitmap graphics 898

graphics for HTML/XML 719

graphics via DPI setting 130, 884

screenshots for WinHelp 872

thumbnail graphics for HTML 605

scheduled operation via **runfm** 979

scope method, WAI

identifying column and row groups 776

identifying columns and rows 776

markup for tables 763, 764

scope, HTML table attribute for WAI 759, 1013

adding via format 768, 769

adding via marker 772

purpose of 763

screen captures, *see* [screenshots](#)

screenshots

converting 899

fuzzy, correcting for HTML 703

scaling, for HTML 606

scaling, for WinHelp 872, 898

script paragraph formats, designating, HTML 650

<script>, HTML paragraph tag 647

scrolling WinHelp topic titles 271

Search, custom marker 833

search, *see* [full-text search](#)

secondary windows

See also [windows, secondary](#)

HTML Help 317

accessing from contents or index 318

accessing from topics 318

defining 317

JavaHelp

size and position settings for 394

using macros for 393

jumping to, in Help systems 224

OmniHelp, specifying jumps to 360

WinHelp

forcing contents to main window 291

specifying 278

section breaks, handling

for print RTF 152

for WinHelp 249

\$_sectionnum, macro variable 801

see-also index entries, *see* <\$nopcode> index entries 444

separator character

between topics, adding 586

in file paths 105

for importing HTML files 446

in system commands 938

SEQ fields, Word, for autonumbers 161

<\$_seqnext>, HTML navigation macro 792

<\$_seqprev>, HTML navigation macro 792

setini.exe, configuration utility 940, 1017

SetPrint, Sundorne Communications plug-in, set by **runfm** 987

setting up a conversion 79

generating and updating 81

importing formats 79

including generated files 81

options for 81

system variables 80

to ASCII DCL 1009

to DITA XML 478, 479

to DocBook XML 559, 560

to Eclipse Help 403, 404

to generic XML 459

to HTML 424, 425

to HTML Help 297, 298

to JavaHelp or Oracle Help 374, 375

to MIF 1006

to OmniHelp 345, 346

to print RTF 146, 195

to WinHelp 243, 244

to XML 425

understanding the process 82

setup, *see* [setting up a conversion](#)

SGML, producing via XML output 458

shading

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- in tables
 - for HTML 745
 - for Word 185
- shed.exe, for WinHelp graphic hotspots 275
- .shg files, WinHelp hypergraphics 275
- _ship, default shipping subdirectory 204, 957
- shipping directory, specifying 975
- sidehead formats
 - converting to HTML 648
 - converting to WinHelp 252
 - converting to Word 159
 - cannot be handled by Word templates 148
 - eliminating via conversion template 68
- sidehead width, specifying for Word 153
- single sourcing, preparing documents for 67, 933
- SIP, Supplementary Ideographic Plane, Unicode 659
- Slovenian font encoding for HTML/XML 431
- small caps, adjusting
 - for WinHelp 254
 - for Word 172
- smart quotes, converting to straight quotes
 - for print RTF 172
 - for WinHelp 256
 - in macros 790
- soft returns in <pre> text, omitting line breaks for 438, 670
- soft returns, *see* forced returns
- solidus, mapped to a forward slash for HTML 662
- sort order, index, specifying 216
 - See also* index, sort order, specifying
- sort strings, index, for HTML-based help 218
- source control, checking files in and out of 979
- space, adding
 - See also* line spacing, adjusting
 - after tables in RTF 185
 - at the end of a macro 789
 - before graphics in HTML 717
 - before tables in HTML 749
 - between topics in a single HTML file 586
- spacer graphic for HTML
 - for indenting images 716
 - for indenting tables 747
- \$\$spacerwidth, macro variable for HTML 793
- spaces
 - around images in HTML table cells,
 - eliminating 717
 - fixed, in Japanese HTML Help 333
 - in configuration settings 104, 113
 - in CSH **newlink** markers, allowing 241
 - in CSS class names, removing or replacing 691
 - in file or path names
 - for commands, double quotes 999, 1001
 - for graphics, eliminating 889
 - for HTML links, avoiding 622
 - not recommended 52, 65, 1032
 - in FrameMaker format names 66
 - in HTML links, removing or replacing 613
 - in marker names, avoiding 140, 837
 - nonbreaking, in Word 164
 - removing from a string value 820
 - thin, adjusting for Word 164
 - trailing, in macros 789
- span class, CSS attribute for character formats 693
- span, HTML table attribute for WAI 832
- special characters, mapping for HTML 660
- special fonts, mapping characters in 662
- specializations
 - handled by **Mif2Go**. 477
- split files
 - See also* split points
 - DITA XML 520
 - HTML 586
 - designating split points for 586
 - determining ObjectIDs of 594
 - naming
 - via custom markers 947
 - via paragraph formats 947
 - suppressing split points for 588
 - titles of, specifying 594
- split points
 - See also* split files
 - for DITA XML files 522
 - for HTML files
 - determining automatically 586
 - for multiple-paragraph headings 590
 - overriding 588
 - preventing dangling headings with 589
 - preventing empty files with 588
 - suppressing 588
 - using Help-contents level numbers for 589
- Split**, custom marker for splitting files 587, 833
- \$_splitid, macro variable 801, 952

ABCDEFGHIJKLMNOPQRSTUVWXYZ

\$\$_splitnum, macro variable 801, 952
 StarOffice, producing RTF for 197
 starting topic, specifying
 for Eclipse Help 405, 413
 for HTML Help 299, 301
 for JavaHelp 376, 382
 for OmniHelp 347, 348
 for Oracle Help 376, 382
 for WinHelp 245, 288
 <\$startrange> marker, *see* index, ranges
 starts, macro string operator 818
 stop words in OmniHelp search 363
 stopping a **Mif2Go** conversion 63
 straddled table columns and rows
 in WinHelp 262
 in Word 185, 186
 strikethrough, as a format override for HTML 657
 string operators for macro expressions, *listed* 813
 stripping paragraph properties for HTML 650
 structure, XML, providing 461
 structured documents
 converting
 to DITA XML 475
 to HTML/XHTML 135
 preparing for conversion 73
 structured documents, converting 73
 style tags, HTML/XML, suppressing in output 648, 653
 \$\$_subsectionnum, macro variable 801
 suffix, file, *see* file, extension
 summary, HTML table attribute for WAI 760, 761, 1013
 Sundorne Communications
 IndexRef plug-in 125, 184, 444
 SetPrint plug-in 987
 Supplementary Ideographic Plane (SIP), Unicode 659
 support for **Mif2Go**, requesting 1029
 symbolic IDs for context-sensitive Help 240
 symbols, converting
 to HTML 662
 to WinHelp 257
 to Word 167
 syntax
 command-line

 for DCL 998
 for **runfm** 980
 configuration-variable assignment 922
 macro variable, for HTML 796
 system
 commands, *see also* commands, system
 commands, to automate conversions 938
 requirements for **Mif2Go** 53
 variables, FrameMaker, converting to text 114
 for HTML 437
 for RTF 157

T

table cells
 HTML, *see* tables: HTML, cells
 rotated, in Word 142, 185
 table of contents, *see* contents
 table structure model, CALS vs. HTML 730
Table*, custom markers for HTML table attributes 833
 TableID
 assigning properties to, for HTML 728
 determining, for HTML 729
 FrameMaker ObjectID for tables 118, 729
 tables, *see*:
 tables: converting
 tables: HTML
 tables: WinHelp
 tables: Word
 tables, list of (LOT)
 See also list of figures or tables, converting
 converting to HTML 444
 converting to HTML Help 325
 converting to Word 181
 preventing conversion of 125
 tables: converting
 to DITA XML 510
 to HTML 727
 to WinHelp 261
 to Word 184
 tables: HTML
 access method, specifying for WAI 764
 adaptive sizing of 742
 attributes
 automatically generated, eliminating 464
 overriding 733, 736, 742
 attributes, specifying

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- precedence of methods 728
- via [Attributes] 736
- via macros 748
- via markers 737
- background color, automatic 739
- border colors 746
- browser-dependent tags for 731
- caption tags 747
- cells
 - attributes of, specifying 751
 - identifying 770, 782
 - rotated, avoiding 426
- color and shading 745
- column groups
 - enumerating 732
 - identifying 731
 - overriding 733
- columns
 - applying CSS class attribute to 732
 - WAI information about 763
- configuration sections subject to override, *listed* 928
- converting to paragraphs 753
- custom ruling and shading 727
- display attributes
 - overriding 736
 - properties for, specifying 735
 - specifying 736
- footer rows, counting 734, 735
- footnotes
 - converting 748
 - positioning 748
- format names, assigning properties to 729
- graphics in, adjusting spacing 717
- groups
 - assigning properties to 729
 - creating 729
 - creating with overrides 930
 - specifying settings for 727
 - using wildcards to specify 730
- header cells, designating 731
- header rows
 - counting 734, 735
 - designating 778
- indenting 747
- line breaks, forcing 744
- list of, converting 444
- macros, specifying 748
- properties of, overriding 928
- properties, assigning 727
- row groups
 - attributes of, specifying 750
 - identifying 731
 - overriding 733
 - specifying 732
- rows
 - attributes of, specifying 737, 751
 - information for WAI 763
 - shading, alternate 745
- ruling properties, converting 727
- space before, adding 749
- splitting files based on 587
- structure, specifying 730
- titles, positioning 747
- variables, FrameMaker, eliminating 748
- WAI markup 759, 1013
 - applying 759
 - method for, choosing 760
 - method for, default, specifying 764
 - overriding 765
 - specifying with custom markers 762
 - strategy for 763
- tables: WinHelp
 - adaptive sizing of 261
 - appearance, adjusting 261
 - converting rows to topics 262
 - titles, positioning 261
- tables: Word
 - cell properties, adjusting 185
 - graphics, repositioning 185
 - in anchored frames, converting 128
 - indents, removing 185
 - landscape, headers and footers 186
 - list of, converting 181
 - rotated cells, avoiding 185
 - space below 185
 - table variables, eliminating 184
 - titles, repositioning 184
- TableSummary**, custom marker for WAI 762, 1013
- TableTitle**, custom marker for WAI 762, 1013
- tabs
 - avoiding in FrameMaker 426
 - converting
 - for print RTF 163
 - for WinHelp 253
 - to spaces for HTML/XML 658, 671
 - eliminating via conversion template 68
 - in autonumbers, eliminating for HTML 649
 - in configuration settings 104

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- replacing with code in HTML/XML 658
- unused, removing for RTF output 164
- tags, HTML/XML, eliminating from output 648, 653
- target frame for HTML jumps 725
- \$\$_tblcols, macro variable 751, 801
- \$\$_tblrows, macro variable 751, 801
- <tbody> elements
 - and RowGroup cells 785
 - overriding [Attributes] for 750
 - required for scope row groups 776
 - tags for HTML tables 732
- technical support for **Mif2Go**, requesting 1029
- templates
 - configuration 849
 - chaining 863
 - creating 861
 - general, organization of 850
 - general, what to include in 862
 - naming convention for 850
 - organization of 849
 - precedence of 863, 919
 - referencing 851
 - DITA <bookmeta> 1039
 - FrameMaker conversion
 - for alternate graphics 69
 - for HTML 426, 866
 - for individual chapter files 68, 865
 - for WinHelp 246
 - for Word 152
 - importing formats from 863, 936
 - specifying at set-up 79
 - troubleshooting 866
 - Mif2Go** configuration 67
 - precedence of 919
 - OmniHelp, modifying 356
 - Word, specifying 148
- temporary configuration overrides 921
- ternary macro operators '?' and ':' 816
- test file title, eliminating 433, 595, 597
- text
 - See also* text insets
 - color, specifying
 - for HTML 669
 - for WinHelp 258
 - for Word 172
 - flows
 - including for HTML 113
 - special, converting to RTF 156
 - for drop-down links, configuring 232
 - frames, using, RTF 152
 - outside a text frame, clipping 902
 - pop-up attributes, HTML Help 306
 - preformatted
 - configuring, HTML 670
 - designating, HTML 646
 - replacing, with code or macros 822
 - underlined, WinHelp 254
 - white, hiding
 - for HTML 652
 - for RTF 173
 - or showing, for callouts 190
 - wrapping for RTF 191
- text insets
 - combining files for HTML 591
 - delimiting, in DITA XML 534
 - for HTML local TOCs 635
 - for WinHelp pop-ups 279
 - specifying links from and to, HTML 624
- <tfoot> elements
 - overriding [Attributes] for 750
 - tags for HTML tables 732
- <th> elements, tags for HTML tables 731
- <thead> elements
 - overriding [Attributes] for 750
 - tags for HTML tables 732
- thin space, adjusting for Word 164
- thumbnails to reference graphics
 - in HTML extracts 604
 - in place of images in HTML 711
- .tif files, exporting 881
- TIFF, graphics export format 130, 884
- Title**, custom marker for split and extract files 597, 833
- title, HTML attribute
 - for images
 - assigning via format 757
 - assigning via marker 1015
 - for links
 - assigning via format 758
 - assigning via marker 757, 1016
 - for tables
 - assigning via marker 1013
 - assigning via TableID 761
 - WAI guidelines for 760
- Title, HTML marker type property 597

ABCDEFGHIJKLMNOPQRSTUVWXYZ

titles

- DITA, alternate, specifying 526
- HTML Help project, specifying 300
- HTML, specifying 433
 - for split and extract files 594
 - to eliminate *Test File* 433, 595, 597
- JavaHelp helpset, specifying 382
- WinHelp
 - file, identifying 288
 - table, repositioning 261
 - topic, configuring 271

.tmb file, Temporary MIF Backup 1032

TOC, *see* contents

toc.xml, Eclipse Help TOC file 412

<\$_top>, HTML navigation macro 635

topic

See also topics

- DITA, starting point of 522
- files, WinHelp
 - assembling contents for 291
 - naming 260
- ID, DITA, specifying 526
- levels in HTML, checking 203
- levels in WinHelp, specifying 289
- starting, specifying
 - for Eclipse Help 405, 413
 - for HTML Help 299, 301
 - for JavaHelp 376, 382
 - for OmniHelp 347, 348
 - for Oracle Help 376, 382
 - for WinHelp 245, 288
- titles in WinHelp, configuring 271
- type, DITA
 - default 525
 - specifying 524

TopicAlias, custom marker for context-sensitive help 833

topics

See also topic

- DITA, *see* DITA, topics
- pop-up, *see* pop-ups; windows, pop-up
- WinHelp
 - adding ALink footnotes to 285
 - converting table rows to 262
 - creating 267

TopicStartCode, custom marker for code at start of topic 792, 833

<\$_TopicStartCode>, predefined macro for

HTML 792

<\$_trail>, predefined macro for HTML 627, 792

trailing space, in macros 789

trails of links, creating for HTML 627

translation, extracting text for 1005

transparency, specifying for graphics 903

trim first, macro string operator 818

trim last, macro string operator 818

truncating cross-reference marker text, WinHelp 261

Turkish

for HTML Help output, specifying 332

for RTF output, specifying 147

Turkish font encoding for HTML/XML 431

Type 11 markers, for WinHelp mid-topic jumps 277

typographic elements

- assigning to a format for DITA output 494
- including for DITA XML 482
- managing in HTML/XML 667
- replacing with other tags 669
- suppressing in HTML/XML 668
 - all 668
 - maintaining overrides 668
 - only in character formats 669
 - only in paragraph formats 668
 - only those used as overrides 668
- use sparingly for DITA XML 493, 495

U

unanchored frames

- converting graphics in, HTML 886
- excluding from HTML output 713
- on master pages, for RTF 885
- processing 126

unattended operation

- designing for 979
- setting up FrameMaker for 980
- using **runfm** for 982

unblocking CHM files 296

underlined text

- as overrides, removing for WinHelp 254
- for hotspots in WinHelp 269, 275
- for links in HTML 609
- solid vs. dotted, for WinHelp hotspots 272
- turning off for tabs in Word 165

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

underscores

- allowed in WinHelp reference strings 270
- avoiding in path and file names 52, 65
 - for HTML Help 622
- disallowed in CSS class names 691
- disallowed to start user variable names 796
- removing from path and file names 1032
- replacing spaces in graphics file names 889

Unicode

- character ranges, assign CSS classes to 694
- characters, mapping in HTML 660
- conversion for HTML 431
- decimal value for character mapping 661
- Mif2Go** support for, understanding 659
- processing for FrameMaker 8 659
- space after, in RTF 148
- values in numeric entity references 658

<Unique ID> tag in MIF 117, 119

UNIX server, relative path to graphics 705

<\$_until>, control structure for macros 815, 816

updating references before converting 126

upper, macro string operator 818

UsePxSuffix, [Graphics] keyword 519

user variables

- FrameMaker, using in macros 802
- Mif2Go**, for run-time values 941

UTF-8 character encoding 460

V

validating HTML documents 431, 453

valign and align, automatically generated, excluding from HTML table cells 464, 739

variable-key configuration sections

- for cross-reference formats, *listed* 928
- for HTML graphics properties, *listed* 930
- for HTML table properties, *listed* 928
- for text formats, *listed* 926
- vs. fixed-key 104

variable-key settings, overriding 925

variables, *see*:

- variables, FrameMaker
- variables, **Mif2Go** configuration
- variables, **Mif2Go** macro
- variables, **Mif2Go** user
- variables, environment

variables, environment

- %OMSYSHOME%, creating 55

variables, FrameMaker

- converting to DITA XML 530
- Running H/F, converting to RTF 156
- system, converting to text 114, 157, 437
- table, eliminating for HTML 748
- user, in macros for HTML 802
- values applied at run time 123

variables, **Mif2Go** configuration

- assigning macros and variables to 923
- assigning values to 922
- capturing settings with 809

variables, **Mif2Go** macro

- See also* macro variables 803
- assigning paragraph content to 803
- assigning values to 797
- incrementing and decrementing 799
- list, using in expressions 818
- list, working with 806
- nesting 798
- predefined
 - all, *listed* 800
 - for extracts, *listed* 603
 - for splits and extracts, *listed* 601
 - starting values for 797

variables, **Mif2Go** macro, predefined

- listed* 800
- uses for 800

variables, **Mif2Go** user

- for run-time values 941
- predefined, in system commands 939

version control, checking files in and out of 979

version of **Mif2Go**

- command-line 995
- how to find 1034

Vista, Microsoft, support for WinHelp 200

\$_volnum, macro variable 801

W

W3C

- HTML 4 specification 429
- placement of <tfoot> elements 733

WAI

- abbr attribute
 - assigning to a paragraph format 768

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- assigning with a special marker 772
- assigning with a special paragraph 772
- attributes
 - assigning to paragraph formats 756, 767
 - assigning values to 769
 - comparing ways to specify 755
 - for links 758
 - image 756
 - image, assigning to a paragraph format 757
 - image, custom markers for 757
 - specifying with paragraph formats 755, 756
 - supplying as paragraph content 756
 - table, custom markers for 762
 - using custom markers for 756
 - using special paragraphs for 772
- axis attribute
 - assigning to a paragraph format 768
 - assigning with a special marker 772
 - assigning with a special paragraph 772
- cells
 - header, group properties of 766
 - identifying 770
 - identifying by row and column 782
 - overriding default settings 784
 - tags for, assigning with paragraph formats 770
- ColGroup cell, *defined* 766
- column groups, defining 766
- guidelines
 - for images 757
 - for links 758
 - for tables 760
- id attribute, assigning with a special marker 772
- id/headers method for table cells 777
- link attributes, assigning to a paragraph format 758
- markup
 - for images 756, 1014
 - for links 758, 1016
 - for tables 759, 1013
- row groups, defining 767
- RowGroup cell, *defined* 767
- scope attribute
 - assigning to a paragraph format 769
 - assigning with a special marker 772
- span attributes 780
- summary attribute 760
- table markup, *see* tables: HTML, WAI markup
- title attribute 760
- warnings, logging as conversion events 115
- wash files via MIF 1005, 1032
- watermark, as background image for HTML 725
- \$_wcount, macro variable 801, 816
- Web Accessibility Initiative, *see* WAI
- Web browsers, *see* browsers
- Web Works Help 200
- WebHelp
 - evaluating 201
 - from HTML Help and RoboHelp 200, 216
- Web-safe colors, *see* colors, Web-safe
- while loops in macros for HTML 816
- <\$while>, control structure for macros 815, 816
- white text
 - hiding
 - for HTML output 652
 - for RTF output 173
 - showing
 - for callouts 190
- whitespace, preserving
 - in DITA block elements 499
 - in HTML output 670
- wildcards, using
 - in configuration settings 106, 113
 - in DCL commands 1001
 - in font names for HTML 664
 - in HTML special-character mappings 660
 - to identify tables for HTML 728
 - to specify table sets for HTML 730
- window
 - browser, opening another 451
 - JavaHelp main, naming 394
 - WinHelp main, naming 291
- Window**, custom marker for HTML Help secondary windows 833
- Windows metafiles (WMFs) 137
 - See also* metafiles; WMF graphics
 - embedding bitmap graphics in 886
- Windows Registry
 - browser command for OmniHelp CSH calls 366
 - CHM files, registering 335
 - key for CLSID for **runfm** 992
- windows, pop-up
 - See also* pop-ups
 - HTML Help 226, 305
 - HTML, require JavaScript 616
 - JavaHelp 226, 394, 396

ABCDEFGHIJKLMNOPQRSTUVWXYZ

- OmniHelp 226, 360
- Oracle Help 226, 399
- WinHelp 273
- windows, secondary
 - See also secondary windows*
 - HTML Help 317
 - defining 317
 - jumping to from a topic 318
 - jumping to from contents or index 318
 - JavaHelp
 - jumping to 399
 - using a macro for 393
 - OmniHelp, jumping to 360
 - Oracle Help, jumping to 399
 - WinHelp
 - jumping to 277
 - not jumping to from contents 291
- winguide.pdf, FDK Platform Guide for Windows 992
- WinHelp
 - advantages and disadvantages of 200
 - compiling
 - for delivery 971
 - from Help Workshop 251
 - via **Mif2Go** 89, 971, 250
 - contents levels, setting 209
 - contents, configuring 288
 - conversion files, location of 1022
 - conversion template for 246
 - elements, identified with character formats 933
 - file name restriction 66
 - files, identifying 288
 - generating 243
 - cross references 259
 - footnotes 258
 - pop-ups from table cells 263
 - reference frames 253
 - run-in headings 252
 - sidehead formats 252
 - special characters 254
 - tables 261
 - topics from table rows 262
 - index entries, maximum length of 212
 - macros, invoking 284
 - overview topic, renaming or eliminating 290
 - platform-specific settings 247
 - project file, naming 248
 - titles, identifying 288
 - topics
 - creating 267
 - from table rows 262
 - starting, specifying 245
 - using for HTML Help pop-ups 307
- WinHelp 2000, producing via WinHelp 4 200
- WinMerge, file comparison tool 60
- WinZip add-on for archiving deliverables 973
- WMF graphics
 - See also metafiles*
 - export format 130, 884
 - for WinHelp
 - files, location of 1022
 - GDI resource problem with 264
 - limitations of 870
- Word
 - bookmarks for every ObjectID 183
 - conversion files, location of 1022
 - cross-reference markers, eliminating 114
 - graphics imported from, extracting 886
 - template, specifying 148
 - using for review 144, 943
 - version 2000, converting to 192
 - version 8, configuring for 149
 - versions of, adjusting for 149
- WordPerfect conversion files, location of 1022
- .wpg files, exporting 881
- wrap
 - and ship conversion output 955
 - text around graphics, for Word 191
 - text lines in <pre> tags for HTML 670
- wrap directory, *see assembly directory*
- _wrap, default assembly directory 204, 957
- wrapping DITA topics in <dita> elements 521
- wzzip.exe
 - for archiving deliverables 973
 - for packaging Eclipse Help topic files 419

X

XHTML

- declaration, suppressing 436
- DocType and DTD 430
- encoding, specifying 432
- for Confluence 4.x, generating 449
- OmniHelp viewer files 369
- tagging for HTML output 428
- using instead of HTML 424

XML

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

- comments, inserting with markers 839
- content type, specifying 461
- file extension, specifying 427, 460
- from unstructured documents 462
- line breaks in, suppressing 437, 461
- links, managing 467
- list types, from unstructured text 466
- output settings, specifying 460
- structure, providing 461
- tag names, deriving from CSS 462
- tags, providing 461
- version, specifying 460
- within HTML 647
- XSLT processing 458

.xml

- default XML file extension 427
- files for JavaHelp, location of 1025

`$_xrefid`, macro variable 615, 801

`<XRefSrcText>` tag in MIF 117, 119

XSLT

- using for XML structure 458
- using to produce DITA XML 474
- using to produce DocBook XML 558

Y

Y: *No entries*

Z

ZIP command for Eclipse Help 419