

---

# DITA2Go User's Guide

For DITA2Go Version 1.2, Update 4

May 15, 2013

Omni Systems, Inc.

---

## Overview

### Lists

§ Contents	1
§ Figures	11
§ Tables	13

### How to set up and use DITA2Go

§ About this guide	17
§1 Getting started with DITA2Go	25
§2 Converting DITA documents	39
§3 Editing configuration files	49
§4 Setting basic conversion options	67
§5 Modifying output appearance	79
§6 Mapping elements to output formats	87
§7 Configuring output formats	109
§8 Configuring format components	141
§9 Specifying conditional processing	161
§10 Including content by reference	175
§11 Defining element sets and properties	179
§12 Creating and deploying user variables	185
§13 Processing related and associative links	189
§14 Generating lists and indexes	197

### Configuring print RTF output

§15 Converting to print RTF	219
-----------------------------	-----

### Configuring on-line Help output

§16 Producing on-line Help	243
§17 Generating WinHelp	281
§18 Generating Microsoft HTML Help	313
§19 Generating OmniHelp	353
§20 Generating JavaHelp or Oracle Help	385
§21 Generating Eclipse Help	413

### Configuring HTML/XML output

§22 Converting to HTML/XHTML	429
§23 Converting to generic XML	449
§24 Converting to DITA XML	455
§25 Configuring DITA maps	493
§26 Converting to DocBook XML	499
§27 Splitting and extracting files	523

§28	Creating HTML links	545
§29	Providing navigation in HTML	555
§30	Mapping text formats to HTML/XML	565
§31	Setting up CSS for HTML	591
§32	Including graphics in HTML	611
§33	Converting tables to HTML	625
<b>Web Accessibility Initiative</b>		
§34	Generating WAI markup for HTML	649
§35	Identifying HTML table structure for WAI	657
§36	Marking HTML table cells for WAI	667
<b>Advanced topics</b>		
§37	Working with macros	679
§38	Working with processing instructions	717
§39	Working with templates	727
§40	Working with graphics	745
§41	Working with content models	753
§42	Overriding configuration settings	765
<b>Project workflow</b>		
§43	Automating DITA2Go conversions	777
§44	Producing deliverable results	787
§45	Converting via DCL	809
§46	Creating a map with DITA2Map	815
<b>Reference</b>		
§A	Technical support for DITA2Go	819
§B	Element type default properties	825
§C	Content model configuration	833
§	Subject index	873

<b>Figures</b>	<b>11</b>
<b>Tables</b>	<b>13</b>
<b>About this guide</b>	<b>17</b>
<b>1 Getting started with DITA2Go</b>	<b>25</b>
1.1 What you need to know . . . . .	25
1.2 What you need to have . . . . .	27
1.3 What you need to do. . . . .	28
1.4 How to update <b>DITA2Go</b> . . . . .	36
1.5 How <b>DITA2Go</b> works . . . . .	37
1.6 How to start and stop <b>DITA2Go</b> . . . . .	38
1.7 How to work with <b>DITA2Go</b> . . . . .	38
1.8 How to uninstall <b>DITA2Go</b> . . . . .	38
<b>2 Converting DITA documents</b>	<b>39</b>
2.1 Creating a <b>DITA2Go</b> conversion project. . . . .	39
2.2 Modifying a <b>DITA2Go</b> conversion project . . . . .	40
2.3 Configuring default <b>DITA2Go</b> project settings. . . . .	40
2.4 Inspecting and editing configuration files . . . . .	44
2.5 Running a <b>DITA2Go</b> conversion. . . . .	44
2.6 Customizing the <b>DITA2Go</b> Project Manager . . . . .	45
2.7 Converting documents from the command line . . . . .	46
<b>3 Editing configuration files</b>	<b>49</b>
3.1 Working with <b>DITA2Go</b> configuration files . . . . .	49
3.2 Editing files with the Configuration Manager . . . . .	49
3.3 Understanding where project settings come from . . . . .	61
3.4 Understanding the rules for configuration settings . . . . .	62
3.5 Specifying file paths in configuration settings. . . . .	64
3.6 Using wildcards in configuration settings . . . . .	65
3.7 Commenting out configuration sections. . . . .	66
3.8 Ending a configuration file. . . . .	66
<b>4 Setting basic conversion options</b>	<b>67</b>
4.1 Specifying operating settings . . . . .	67
4.2 Logging conversion events. . . . .	74
4.3 Identifying files and elements . . . . .	76
4.4 Processing graphics . . . . .	77
<b>5 Modifying output appearance</b>	<b>79</b>
5.1 Understanding where to modify formats . . . . .	79
5.2 Understanding how to modify formats. . . . .	80

5.3	Changing how the output looks . . . . .	80
5.4	Determining how an element is rendered . . . . .	83
<b>6</b>	<b>Mapping elements to output formats</b>	<b>87</b>
6.1	Understanding how to assign formats. . . . .	87
6.2	Specifying options for naming formats. . . . .	87
6.3	Mapping outputclass attribute values to formats . . . . .	89
6.4	Mapping element paths to output formats . . . . .	91
6.5	Mapping element attributes to output formats . . . . .	95
6.6	Specifying formats for cross references . . . . .	101
6.7	Specifying formats for footnotes . . . . .	102
6.8	Specifying options for figures. . . . .	102
6.9	Specifying formats and options for tables . . . . .	103
6.10	Specifying options for special lists . . . . .	104
6.11	Specifying options for draft comments. . . . .	105
6.12	Specifying options for maps . . . . .	105
6.13	Deciding where to display title and shortdesc . . . . .	107
<b>7</b>	<b>Configuring output formats</b>	<b>109</b>
7.1	Understanding the purpose of output formats. . . . .	109
7.2	Working with format configuration files . . . . .	110
7.3	Creating aliases to existing format names . . . . .	112
7.4	Understanding how to define output formats . . . . .	114
7.5	Understanding text output formats . . . . .	119
7.6	Configuring text output formats . . . . .	121
7.7	Configuring table output formats . . . . .	129
7.8	Configuring page layouts for RTF output. . . . .	134
7.9	Inserting line, column, and page breaks in output . . . . .	138
<b>8</b>	<b>Configuring format components</b>	<b>141</b>
8.1	Managing format components . . . . .	141
8.2	Defining border format components. . . . .	144
8.3	Defining shading format components. . . . .	145
8.4	Overriding border and shading properties . . . . .	145
8.5	Configuring output numbering properties . . . . .	146
8.6	Configuring run-in headings for text formats. . . . .	153
8.7	Defining cross-reference output formats . . . . .	155
8.8	Configuring trademark formats. . . . .	157
8.9	Localizing output headings, labels, and names. . . . .	157
<b>9</b>	<b>Specifying conditional processing</b>	<b>161</b>
9.1	Extracting conditions from ditaval files . . . . .	161
9.2	Defining conditional actions. . . . .	162
9.3	Including flags for ditaval conditions . . . . .	165
9.4	Configuring conditional flags . . . . .	166



9.5	Assigning attributes with conditional flags . . . . .	169
9.6	Scoping and filtering within maps . . . . .	169
<b>10</b>	<b>Including content by reference</b>	<b>175</b>
10.1	Pushing and pulling content by reference . . . . .	175
10.2	Referencing external code or text fragments . . . . .	176
<b>11</b>	<b>Defining element sets and properties</b>	<b>179</b>
11.1	Defining sets of elements . . . . .	179
11.2	Specifying properties of element types . . . . .	179
<b>12</b>	<b>Creating and deploying user variables</b>	<b>185</b>
12.1	Understanding how <b>DITA2Go</b> user variables work . . . . .	185
12.2	Assigning variable names to element paths . . . . .	186
12.3	Including user variables in <b>DITA2Go</b> macros. . . . .	186
12.4	Deploying user variables in template macros. . . . .	187
<b>13</b>	<b>Processing related and associative links</b>	<b>189</b>
13.1	Understanding how <b>DITA2Go</b> treats retables . . . . .	189
13.2	Generating and including related links. . . . .	189
13.3	Appending links to topics. . . . .	190
13.4	Including descriptions with related links . . . . .	191
13.5	Generating associative links for Help output . . . . .	192
13.6	Formatting links in output . . . . .	192
13.7	Changing link path for peer related links. . . . .	196
<b>14</b>	<b>Generating lists and indexes</b>	<b>197</b>
14.1	Understanding how <b>DITA2Go</b> produces lists . . . . .	197
14.2	Naming generated HTML list and index files . . . . .	198
14.3	Generating a table of contents . . . . .	198
14.4	Generating a list of figures. . . . .	201
14.5	Generating a list of tables. . . . .	202
14.6	Treating figure titles as table titles . . . . .	203
14.7	Producing a glossary . . . . .	203
14.8	Producing an index. . . . .	206
14.9	Configuring variant booklist components . . . . .	213
<b>15</b>	<b>Converting to print RTF</b>	<b>219</b>
15.1	Setting up a print RTF project . . . . .	219
15.2	Adjusting output for different versions of Word . . . . .	224
15.3	Converting paragraph and character formats . . . . .	225
15.4	Modifying text appearance. . . . .	227
15.5	Converting cross references and hypertext links . . . . .	229
15.6	Converting tables to print RTF. . . . .	232
15.7	Managing graphics for print RTF. . . . .	234
15.8	Including RTF code for Word output. . . . .	238

15.9	Turning on revision tracking in Word .....	239
15.10	Managing Word output after conversion .....	239
15.11	Converting to OpenOffice or StarOffice. ....	241
<b>16</b>	<b>Producing on-line Help</b>	<b>243</b>
16.1	Weighing Help-system alternatives .....	243
16.2	Completing Help system construction .....	247
16.3	Producing contents and index for Help systems. ....	248
16.4	Configuring contents entries for Help systems. ....	250
16.5	Configuring index entries for Help systems .....	251
16.6	Providing related-topic links for Help systems. ....	258
16.7	Jumping to secondary windows in Help systems .....	262
16.8	Creating pop-up topics for Help systems .....	263
16.9	Including expandable sections in Help topics. ....	264
16.10	Setting up Context Sensitive Help (CSH). ....	277
16.11	Setting up a dynamic modular Help system .....	280
<b>17</b>	<b>Generating WinHelp</b>	<b>281</b>
17.1	Obtaining tools for WinHelp .....	281
17.2	Setting up a WinHelp project .....	281
17.3	Converting text .....	286
17.4	Converting cross references .....	288
17.5	Converting tables to WinHelp RTF .....	290
17.6	Managing graphics for WinHelp .....	292
17.7	Configuring WinHelp topics. ....	294
17.8	Creating jumps and pop-ups for WinHelp .....	299
17.9	Invoking WinHelp macros .....	302
17.10	Creating related-topic links in WinHelp. ....	303
17.11	Configuring index entries for WinHelp .....	305
17.12	Configuring contents for WinHelp .....	306
17.13	Creating browse sequences. ....	310
<b>18</b>	<b>Generating Microsoft HTML Help</b>	<b>313</b>
18.1	Understanding how <b>DITA2Go</b> produces HTML Help .....	313
18.2	Understanding why Unicode is not the answer. ....	314
18.3	Setting up an HTML Help project .....	315
18.4	Customizing HTML Help display features. ....	319
18.5	Creating pop-ups for HTML Help .....	322
18.6	Creating links and hypertext jumps in HTML Help. ....	323
18.7	Creating related-topic links for HTML Help .....	325
18.8	Using secondary windows in HTML Help. ....	332
18.9	Generating contents and index for HTML Help. ....	334
18.10	Providing full-text search (FTS) for HTML Help .....	339
18.11	Setting up CSH for HTML Help. ....	340

18.12	Generating HTML Help in non-Western languages	344
18.13	Compiling and testing HTML Help	346
18.14	Mapping and merging CHM files	348
<b>19</b>	<b>Generating OmniHelp</b>	<b>353</b>
19.1	Understanding how OmniHelp works	353
19.2	Setting up OmniHelp viewer control files	354
19.3	Setting up an OmniHelp project	357
19.4	Using CSS with OmniHelp	361
19.5	Customizing OmniHelp display features	363
19.6	Choosing navigation features for OmniHelp	367
19.7	Configuring contents and index for OmniHelp	367
19.8	Providing related-topic links in OmniHelp	370
19.9	Jumping to secondary windows in OmniHelp	370
19.10	Configuring full-text search for OmniHelp	371
19.11	Setting up CSH for OmniHelp	375
19.12	Merging OmniHelp projects	377
19.13	Assembling OmniHelp files for viewing	380
19.14	Deploying OmniHelp	381
<b>20</b>	<b>Generating JavaHelp or Oracle Help</b>	<b>385</b>
20.1	Deciding which Java Help system to use	385
20.2	Obtaining tools for a Java-based Help system	385
20.3	Setting up a JavaHelp or Oracle Help project	386
20.4	Generating contents and index	395
20.5	Providing full-text search for JavaHelp / Oracle Help	397
20.6	Creating and viewing a Java Archive (JAR) file	400
20.7	Converting a glossary to JavaHelp 2	401
20.8	Defining windows for JavaHelp or Oracle Help	403
20.9	Linking to destinations within topics	409
20.10	Creating ALinks for Oracle Help	409
20.11	Merging JavaHelp or Oracle Help systems	410
20.12	Setting up CSH for JavaHelp or Oracle Help	410
<b>21</b>	<b>Generating Eclipse Help</b>	<b>413</b>
21.1	Understanding how Eclipse Help works	413
21.2	Setting up an Eclipse Help project	413
21.3	Configuring Eclipse Help manifest files	416
21.4	Configuring contents and index for Eclipse Help	420
21.5	Configuring search properties for Eclipse Help	423
21.6	Merging Eclipse Help projects	423
21.7	Setting up CSH for Eclipse Help	425
21.8	Packaging Eclipse Help files	427

<b>22</b>	<b>Converting to HTML/XHTML</b>	<b>429</b>
22.1	Deciding which type of output to produce . . . . .	430
22.2	Setting up an HTML project. . . . .	430
22.3	Including starting code and entity references . . . . .	432
22.4	Supplying values for the <head> element. . . . .	432
22.5	Specifying HTML <body> attributes . . . . .	438
22.6	Specifying document-wide properties for HTML . . . . .	438
22.7	Defining text colors for HTML. . . . .	439
22.8	Importing HTML files as insets . . . . .	441
22.9	Providing hover text for links in HTML. . . . .	441
22.10	Generating XHTML for Confluence 4.x . . . . .	442
22.11	Exporting content for database input . . . . .	443
22.12	Specifying a starting topic for HTML or XHTML. . . . .	443
22.13	Using framesets. . . . .	443
22.14	Passing W3C validation tests . . . . .	445
<b>23</b>	<b>Converting to generic XML</b>	<b>449</b>
23.1	Setting up a generic XML project. . . . .	449
23.2	Specifying generic XML output settings . . . . .	449
23.3	Providing XML tags and structure . . . . .	451
23.4	Configuring links for generic XML . . . . .	454
<b>24</b>	<b>Converting to DITA XML</b>	<b>455</b>
24.1	Generating DITA XML output from DITA input. . . . .	455
24.2	Setting up a DITA XML project. . . . .	455
24.3	Specifying general options for DITA . . . . .	458
24.4	Configuring DITA elements . . . . .	459
24.5	Nesting DITA block elements . . . . .	471
24.6	Specifying options for tables in DITA XML . . . . .	480
24.7	Specifying options for images in DITA XML . . . . .	482
24.8	Configuring DITA topics . . . . .	484
24.9	Configuring cross references and links for DITA. . . . .	489
24.10	Including CSH targets in DITA XML . . . . .	491
24.11	Overriding DITA settings with markers . . . . .	491
<b>25</b>	<b>Configuring DITA maps</b>	<b>493</b>
25.1	Configuring ditamaps . . . . .	493
25.2	Overriding map settings with PI markers . . . . .	498
<b>26</b>	<b>Converting to DocBook XML</b>	<b>499</b>
26.1	Generating DocBook XML with <b>DITA2Go</b> . . . . .	499
26.2	Setting up a DocBook XML project. . . . .	500
26.3	Specifying general options for DocBook . . . . .	502
26.4	Configuring DocBook elements . . . . .	504
26.5	Nesting DocBook block elements. . . . .	511

26.6	Designating ancestors for table elements . . . . .	519
26.7	Specifying options for figure elements . . . . .	520
26.8	Overriding DocBook settings with PI markers . . . . .	521
<b>27</b>	<b>Splitting and extracting files</b>	<b>523</b>
27.1	Splitting and extracting vs. chunking . . . . .	523
27.2	Chunking DITA maps . . . . .	523
27.3	Splitting files . . . . .	526
27.4	Extracting files . . . . .	528
27.5	Identifying split and extract files . . . . .	530
27.6	Inserting HTML code in split and extract files . . . . .	534
27.7	Referencing split and extract files . . . . .	536
27.8	Customizing and replacing extracts . . . . .	537
<b>28</b>	<b>Creating HTML links</b>	<b>545</b>
28.1	Understanding sources of links . . . . .	545
28.2	Specifying link appearance . . . . .	545
28.3	Specifying link destination . . . . .	549
28.4	Creating jumps to particular windows for HTML . . . . .	550
28.5	Converting DITA cross-reference links to HTML . . . . .	551
28.6	Linking to other files and other <b>DITA2Go</b> projects . . . . .	553
28.7	Linking to external destinations . . . . .	554
<b>29</b>	<b>Providing navigation in HTML</b>	<b>555</b>
29.1	Understanding how navigation links work . . . . .	555
29.2	Generating trails of links . . . . .	555
29.3	Creating a browse sequence . . . . .	559
<b>30</b>	<b>Mapping text formats to HTML/XML</b>	<b>565</b>
30.1	Choosing how to map formats . . . . .	565
30.2	Mapping paragraph formats . . . . .	566
30.3	Mapping character formats . . . . .	569
30.4	Assigning properties to text formats . . . . .	570
30.5	Mapping special characters . . . . .	574
30.6	Mapping fonts . . . . .	576
30.7	Managing typographic elements for HTML or XML . . . . .	579
30.8	Specifying text colors for HTML . . . . .	580
30.9	Configuring preformatted text for HTML/XML . . . . .	581
30.10	Converting footnotes to HTML or XML . . . . .	581
30.11	Converting list formats to HTML ( <i>deprecated</i> ) . . . . .	584
<b>31</b>	<b>Setting up CSS for HTML</b>	<b>591</b>
31.1	Deciding whether to use CSS . . . . .	591
31.2	Understanding how to use CSS . . . . .	591
31.3	Understanding how <b>DITA2Go</b> generates CSS . . . . .	592
31.4	Specifying CSS file and link options . . . . .	593

31.5	Understanding how CSS affects other options . . . . .	596
31.6	Linking to alternate CSS files . . . . .	597
31.7	Assigning CSS classes . . . . .	599
31.8	Customizing CSS properties . . . . .	606
<b>32</b>	<b>Including graphics in HTML</b>	<b>611</b>
32.1	Locating graphics files for HTML . . . . .	611
32.2	Specifying options for HTML graphics . . . . .	612
32.3	Omitting graphics from HTML output . . . . .	613
32.4	Selecting and modifying graphics . . . . .	613
32.5	Positioning graphics in HTML output . . . . .	617
32.6	Specifying HTML image attributes . . . . .	619
32.7	Providing (or omitting) alternate text for images . . . . .	620
32.8	Scaling images for HTML . . . . .	620
32.9	Creating image maps for HTML . . . . .	622
32.10	Supplying a background image or watermark . . . . .	624
<b>33</b>	<b>Converting tables to HTML</b>	<b>625</b>
33.1	Assigning properties to tables . . . . .	625
33.2	Defining sets of tables . . . . .	626
33.3	Specifying table structure . . . . .	627
33.4	Specifying table attributes . . . . .	632
33.5	Positioning tables, table titles, and table footnotes . . . . .	640
33.6	Using macros to control table properties . . . . .	642
33.7	Converting tables to paragraphs . . . . .	647
<b>34</b>	<b>Generating WAI markup for HTML</b>	<b>649</b>
34.1	Comparing DITA2Go markup methods for WAI . . . . .	649
34.2	Applying WAI markup to images . . . . .	650
34.3	Applying WAI markup to links . . . . .	651
34.4	Applying WAI markup to tables . . . . .	652
<b>35</b>	<b>Identifying HTML table structure for WAI</b>	<b>657</b>
35.1	Identifying table rows and columns . . . . .	657
35.2	Associating table cells with header cells . . . . .	660
<b>36</b>	<b>Marking HTML table cells for WAI</b>	<b>667</b>
36.1	Understanding table cell settings . . . . .	667
36.2	Using the scope method to identify table cells . . . . .	667
36.3	Using the id/headers method to identify table cells . . . . .	669
36.4	Overriding default table-cell settings . . . . .	675
36.5	Using ColGroup and RowGroup cells . . . . .	676
<b>37</b>	<b>Working with macros</b>	<b>679</b>
37.1	Defining and invoking macros . . . . .	679
37.2	Accessing DITA2Go macro libraries . . . . .	684

37.3 Using macro variables .....	687
37.4 Using multiple-value list variables .....	695
37.5 Accessing settings with configuration macros .....	699
37.6 Using expressions in macros .....	700
37.7 Passing a parameter to a macro .....	709
37.8 Debugging macros .....	709
37.9 Deploying macros and macro variables .....	710
37.10 Using macros to fine-tune HTML or XML output .....	713
<b>38 Working with processing instructions</b>	<b>717</b>
38.1 Understanding <b>DITA2Go</b> PI markers .....	717
38.2 Understanding effects of PI markers .....	718
38.3 Adding attributes with PI markers .....	721
38.4 Assigning properties to PI marker types .....	723
38.5 Inserting code with PI markers .....	724
<b>39 Working with templates</b>	<b>727</b>
39.1 Working with configuration templates .....	727
39.2 Referencing configuration files and templates .....	731
39.3 Including document-specific configuration files .....	732
39.4 Deciding which configuration file to edit .....	734
39.5 Creating your own configuration templates .....	741
<b>40 Working with graphics</b>	<b>745</b>
40.1 Choosing an appropriate graphics format .....	745
40.2 Replacing and relocating graphics files .....	746
40.3 Specifying custom settings for individual graphics .....	752
<b>41 Working with content models</b>	<b>753</b>
41.1 Understanding <b>DITA2Go</b> content models .....	753
41.2 Modifying or replacing a content model .....	753
41.3 Preparing a content model for use with <b>DITA2Go</b> .....	755
41.4 Understanding content-model configurations .....	756
41.5 Understanding how <b>DITA2Go</b> uses content models .....	759
41.6 Inspecting and correcting element types .....	760
41.7 Specializing or modifying DITA topic types .....	761
41.8 Extracting content-model debug information .....	764
<b>42 Overriding configuration settings</b>	<b>765</b>
42.1 Using a different configuration for selected files .....	765
42.2 Overriding settings with PI markers or macros .....	766
42.3 Overriding configuration settings with text .....	776
<b>43 Automating DITA2Go conversions</b>	<b>777</b>
43.1 Executing operating-system commands .....	777
43.2 Converting autonumbers for database systems .....	780

43.3 Renaming output files for automated systems . . . . .	781
<b>44 Producing deliverable results</b>	<b>787</b>
44.1 Understanding <b>DITA2Go</b> pre- and post-processing. . . . .	787
44.2 Activating and logging production of deliverables. . . . .	788
44.3 Understanding path values for deliverables . . . . .	788
44.4 Clearing out old files before converting . . . . .	789
44.5 Gathering additional files before converting . . . . .	791
44.6 Assembling files for distribution . . . . .	792
44.7 Placing graphics files for distribution. . . . .	796
44.8 Placing CSS or XSL files for assembly . . . . .	800
44.9 Gathering files for an HTML project: an example . . . . .	801
44.10 Gathering and processing Help-system files. . . . .	802
44.11 Archiving deliverables . . . . .	803
44.12 Placing deliverables in a shipping directory . . . . .	806
44.13 Postprocessing separately from converting . . . . .	807
<b>45 Converting via DCL</b>	<b>809</b>
45.1 How the DCL filter works . . . . .	809
45.2 Using the DCL filter . . . . .	809
45.3 DCL command-line syntax. . . . .	810
45.4 Specifying output file paths and names . . . . .	812
45.5 About DCL technology. . . . .	813
<b>46 Creating a map with DITA2Map</b>	<b>815</b>
46.1 Understanding how <b>DITA2Map</b> works . . . . .	815
46.2 Setting up a <b>DITA2Map</b> project . . . . .	815
46.3 Specifying <b>DITA2Map</b> configuration options. . . . .	815
46.4 Running <b>DITA2Map</b> . . . . .	817
<b>A Technical support for DITA2Go</b>	<b>819</b>
<b>B Element type default properties</b>	<b>825</b>
<b>C Content model configuration</b>	<b>833</b>
<b>RTF keyword index</b>	<b>837</b>
<b>HTML/XML keyword index</b>	<b>849</b>
<b>Subject index</b>	<b>873</b>



Figure 18-1 HTML Help Workshop Project tab .....	320
Figure 18-2 HTML Help Workshop Window Types.....	321
Figure 22-1 RGB color 0099CC .....	440
Figure 36-1 Extent of row and column groups.....	672
Figure 36-2 Extent of column and row spans.....	673
Figure 36-3 Extent of column and row IDs .....	675



Table 0-1 <b>DITA2Go User's Guide</b> formats and archives. . . . .	17
Table 2-1 Configuration Wizard output-specific settings . . . . .	42
Table 3-1: Absolute vs. relative file-path settings . . . . .	65
Table 4-1 Output types, file extensions, project configuration files . . . . .	71
Table 4-2: Topics included in output based on PrintProject setting. . . . .	72
Table 7-1 Valid suffixes for names of formats and format components . . . . .	114
Table 7-2 Inline properties for text formats . . . . .	123
Table 7-3 Block properties for output paragraph formats . . . . .	124
Table 7-4 Border properties for paragraph formats . . . . .	128
Table 7-5 Table output format properties. . . . .	130
Table 7-6 Format properties of table rows . . . . .	132
Table 7-7 Format properties of table cells . . . . .	133
Table 7-8 RTF output section properties . . . . .	135
Table 8-1 Properties of border format components . . . . .	144
Table 8-2 Properties of number streams. . . . .	147
Table 8-3 Properties of number format components . . . . .	148
Table 8-4 Building blocks for run-in heading formats. . . . .	154
Table 8-5 Building blocks for cross-reference formats . . . . .	155
Table 8-6 Default cross-reference format names and definitions . . . . .	157
Table 9-1 Text properties for flags. . . . .	166
Table 10-1 <b>DITA2Go</b> support for push and pull conrefs . . . . .	175
Table 11-1 Element type properties . . . . .	181
Table 13-1 Default treatment of related links by link source and type . . . . .	189
Table 14-1: Default file name suffixes for generated files. . . . .	198
Table 15-1 RTF differences between Word 7/95 and later versions . . . . .	224
Table 15-2 Effects of cross-reference settings in Word. . . . .	230
Table 15-3 Graphics scale percentages for Word versions . . . . .	236
Table 16-1 Index link options for KeywordRefs in HTML-based Help . . . . .	255
Table 16-2 Effects of drop-down format properties. . . . .	267
Table 17-1 Starting and following format properties for topics and hotspots . . . . .	295
Table 17-2 Effects of format properties on topics and hotspots . . . . .	296
Table 18-1 ALink and KLink jump properties for HTML Help . . . . .	327
Table 18-2 Binary TOC/Index advantages and disadvantages for HTML Help . . . . .	336
Table 18-3 Rationale for HHW settings by CHM role . . . . .	351
Table 18-4 HTML Help Workshop settings for stand-alone vs. merged CHMs . . . . .	352
Table 19-1 OmniHelp viewer control files included in the distribution . . . . .	356
Table 19-2 OmniHelp data and control files generated by <b>DITA2Go</b> . . . . .	357
Table 19-3 OmniHelp navigation features. . . . .	367
Table 19-4 OmniHelp viewer files copied from OHViewPath to WrapPath. . . . .	380
Table 20-1 [JavaHelpOptions] pop-up and secondary window properties . . . . .	405

Table 20-2	[ <i>JavaHelp window name</i> ] window-access object properties. . . . .	406
Table 20-3	Oracle Help for Java window properties . . . . .	408
Table 21-1:	Eclipse Help properties in either MANIFEST.MF or plugin.xml. . . . .	416
Table 22-1	Ways to express Web-safe RGB color values . . . . .	441
Table 22-2	Default options for Confluence 4.x XHTML. . . . .	442
Table 22-3	Characters replaced or removed for W3C validation. . . . .	446
Table 24-1	Precedence of DITA topic type assignment methods. . . . .	486
Table 24-2	Predefined marker types for DITA XML. . . . .	492
Table 25-1	DITA map navigation elements from PI markers . . . . .	498
Table 25-2	Predefined PI marker types for DITA maps and bookmarks. . . . .	498
Table 26-1	Predefined PI marker types for DocBook . . . . .	521
Table 27-1	Precedence of HTML page titles . . . . .	531
Table 27-2	Extract code insertion methods . . . . .	535
Table 27-3	Basic macro-insertion keywords and locations . . . . .	535
Table 27-4	Keyword prefixes for split or extract code insertion . . . . .	535
Table 27-5	Code insertion keywords for split and extract files . . . . .	536
Table 27-6	Predefined macro variables for splits and extracts. . . . .	537
Table 27-7	Predefined PI marker types for extracts . . . . .	537
Table 27-8	Predefined macro variables for extract replacement code . . . . .	539
Table 29-1	Default destination and label values for browse macros . . . . .	560
Table 29-2	Component macro variables for browse macros . . . . .	560
Table 29-3	Scope of [NavigationMacros] keywords . . . . .	563
Table 29-4	Default values of text-link browse keywords. . . . .	563
Table 29-5	Default values of button browse keywords . . . . .	564
Table 30-1	HTML properties for paragraph and character formats . . . . .	571
Table 30-2	Special characters to replace for HTML/XML output. . . . .	575
Table 31-1	Default CSS file options when [HtmlOptions]Stylesheet is used . . . . .	596
Table 31-2	CSS-dependent default values of options. . . . .	597
Table 33-1	Precedence of table and cell property assignment methods. . . . .	626
Table 33-2	Browser-dependent HTML tags for tables. . . . .	628
Table 33-3	Default counts of table header rows/columns and footer rows . . . . .	631
Table 35-1	Format properties for WAI table-cell attributes. . . . .	662
Table 35-2	Using paragraph formats to identify table cells (example) . . . . .	664
Table 36-1	WAI scope attributes for table cells . . . . .	667
Table 36-2	WAI id/header table cell attributes. . . . .	670
Table 36-3	ColGroup property effects . . . . .	677
Table 36-4	RowGroup property effects . . . . .	677
Table 37-1	Predefined macros for HTML output. . . . .	683
Table 37-2	Predefined macros for RTF output. . . . .	684
Table 37-3	Dita2Go macro libraries . . . . .	684
Table 37-4	Character literals for macro variables. . . . .	689
Table 37-5	Predefined macro variables . . . . .	691
Table 37-6	Operators for HTML macro expressions . . . . .	701

Table 37-7	Format components for displaying expression results . . . . .	703
Table 37-8	Predefined control-structure elements . . . . .	704
Table 37-9	String operators in macro expressions . . . . .	707
Table 37-10	Macro code placement properties . . . . .	712
Table 38-1	PI marker types with predefined effects . . . . .	719
Table 38-2	Elements to which attribute PI markers apply, by output type . . . . .	722
Table 38-3	Effects of [MarkerTypes] properties . . . . .	723
Table 39-1	Output-type-specific general configuration files . . . . .	728
Table 39-2	Configuration options determined at run time . . . . .	733
Table 39-3	Intended scope of settings by configuration type . . . . .	735
Table 39-4	Chain of general configuration files for HTML Help output . . . . .	736
Table 39-5	Output types and starting project configuration files . . . . .	737
Table 39-6	Editable local output-specific configuration files . . . . .	738
Table 39-7	Output types and format configuration files . . . . .	739
Table 39-8	Language configuration files . . . . .	740
Table 39-9	Macro configuration files . . . . .	741
Table 40-1	RTF replacement graphics file mappings and locations . . . . .	749
Table 41-1	Configuration files for <b>DITA2Go</b> built-in content models . . . . .	754
Table 42-1	Precedence of settings in configuration files and templates . . . . .	766
Table 42-2	Fixed-key configuration sections subject to overrides . . . . .	770
Table 42-3	Text configuration sections subject to overrides . . . . .	772
Table 42-4	Cross-reference sections subject to overrides . . . . .	773
Table 42-5	HTML table sections subject to overrides . . . . .	774
Table 42-6	HTML graphic sections subject to overrides . . . . .	775
Table 44-1	Default files copied from project directory to wrap directory . . . . .	794
Table 44-2	Files copied by default to the wrap directory . . . . .	795
Table 44-3	Default graphics files copied for assembly . . . . .	797
Table 44-4	Automation settings activated by CompileHelp or FTSCCommand . . . . .	803
Table 44-5	Default base file name for deliverables archive . . . . .	806
Table 45-1	DCL intermediate input and output options . . . . .	812
Table A-1	Examples of build numbers for <b>DITA2Go</b> DLL files . . . . .	820
Table B-1	Default properties assigned to elements . . . . .	825



# About this guide

The **DITA2Go User's Guide** describes how to install and use Omni Systems **DITA2Go** version 1.2 software and configuration files, to convert DITA documents to any of several output types. This guide assumes you are familiar with DITA, and also with the output type to which you are converting DITA files. **DITA2Go** supports the DITA 1.2 specification.

*In this section:*

§ [Availability](#) on page 17

§ [New information](#) on page 17

§ [Colophon](#) on page 22

## Availability

The **DITA2Go User's Guide** is available in the formats listed in [Table 0-1](#). All editions except the PDF edition are produced with **DITA2Go**.

**Table 0-1 DITA2Go User's Guide formats and archives**

Format	Archive	Starting file	Comments*
Eclipse Help	UGDITA2Go_EH4.zip	<i>Not applicable</i>	Requires Eclipse platform or infocenter
HTML	UGDITA2Go_HTM4.zip	_ugdita2go.htm	Requires JavaScript to enable CSS
HTML Help	UGDITA2Go_HH4.zip	ugdita2go.chm	Must be registered for network use
JavaHelp 2	UGDITA2Go_JH4.zip	ugdita2go.jar	Requires Java Runtime Environment
OmniHelp	UGDITA2Go_OH4.zip	_ugdita2go.htm	Requires browser and JavaScript
Oracle Help	UGDITA2Go_OHJ4.zip	ugdita2go.hs	Requires Java Virtual Machine
PDF	UGDITA2Go_PDF4.zip	ugdita2go.pdf	Designed for duplex printing
Word 2007	UGDITA2Go_RTF4.zip	ugdita2go.rtf	Includes active cross references and hypertext links
XHTML	UGDITA2Go_XH4.zip	_ugdita2go.htm	Requires JavaScript for CSS; some browsers ignore

\*See § [Colophon](#) on page 22 for additional information.

You can download any of these archives from Omni Systems:

<http://www.dita2go.com>

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

## New information

The **DITA2Go 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 1.2, update 4

**Editing configuration files:**

- **Edit settings with the DITA2Go Configuration Manager.** New tool, accessible from the Project Manager; see §3.2 [Editing files with the Configuration Manager](#) on page 49.

- **Use relative path settings with care!** Relativity of some path settings has changed, notably those listed in [Table 3-1](#); see §3.5 [Specifying file paths in configuration settings](#) on page 64.

#### Setting basic conversion options:

- **One setting for output file extension.** Use `[Setup]FileSuffix` for all output types; see §4.1.6 [Checking output type and file extension](#) on page 70. 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 undefined formats.** New setting `ShowUndefinedFormats`, on by default; see §4.2 [Logging conversion events](#) on page 74.

#### Configuring output formats:

- **Format names require valid suffixes.** Not just a convention any more; see §7.4 [Understanding how to define output formats](#) on page 114.

#### Configuring format components:

- **Stream numbers have changed for figures, tables, and equations.** If you number figures, tables, or equations, and you use other than the default subformat definitions supplied in system configuration files, you must change your subformat definitions so that figures use stream number 8, tables use stream number 9, and equations use stream number 10; see §8.5.2 [Defining number streams](#) on page 147.

#### Generating lists and indexes:

- **You get lists, like it or not, as per DITA spec.** DITA2Go is required to output whatever booklist components are present in your bookmap; see §14.1 [Understanding how DITA2Go produces lists](#) on page 197.
- **Generate and configure a glossary.** New settings `GenerateGLS`, `GLSFile`, `GLSSuffix`, `GLSTitle`, `GLSFormat`, `GlossaryItem`, `GLSTitleFormat`, `GLSXrefFormat`, `ResetAbbrevAt`, `UseAbbrevInTitles`; see §14.7 [Producing a glossary](#) on page 203.
- **Specify `IDXTITLE` in a language configuration file (*correction*).** This setting belongs in section `[IndexText]`, not in `[Index]`; see §14.8.1 [Specifying output formats for the index](#) on page 207.
- **RTF indexes now have active links.** DITA2Go uses a new method that does not involve creating the Word `INDEX` field. The following settings, which have no reasonable equivalents in the new system, are deprecated: `IndexStyle`, `IndexLeader`, and `IndexColumns`; see §14.8.4 [Configuring index references](#) on page 209.
- **Override default formats for index entries and references.** New sections `IndexEntryFormats` and `IndexRefParaFormats`; see §14.8.2 [Overriding formats for index entries and references](#) on page 207.
- **Configure *see* and *see-also* entries for variant indexes.** New `[IndexSeeText]` settings `SeeStartListIndex` and `SeeAlsoStartListIndex`, new `[IndexSeeFormats]` settings `IndexSeeListStart` and `IndexSeeAlsoListStart`; see §14.8.3 [Configuring \*see\* and \*see-also\* index entries](#) on page 208.
- **Configure index references.** New settings `UseCompactForm`, `FullIndexRanges`, `IndexRefSep`, and `IndexRangeSep`; see §14.8.4 [Configuring index references](#) on page 209.



- **Configure index alphabetic headings and links.** New settings `IndexLettersFormat` and `IndexTopLettersFormat`; see §14.8.5 [Including heading letters in the index](#) on page 210.
- **Toss `[HTMLOptions]UseIndex`.** This setting is no longer needed for a non-Help HTML index, which will be built if you include `<indexlist>` in your map; see §14.8.6 [Configuring index features for HTML output](#) on page 211.
- **Generate multiple variants of the same booklist type.** New sections `[BookLists]`, `[*BList]`, `[*BLItems]`, `[*BLForms]`, `[*BLRefForms]`, `[IndexClasses]`, `[IndexLists]`; see §14.9 [Configuring variant booklist components](#) on page 213.
- **Provide outputclass via PI for indexterms.** If your bookmap contains more than one `indexlist`; see §14.9.5 [Mapping indexterms to variant indexes](#) on page 217.

#### ***Producing on-line Help:***

- **Use `<data />` elements for CSH targets.** Additional support for context-sensitive help; see §16.10.2 [Specifying CSH mappings](#) on page 278.

#### ***Generating Microsoft HTML Help:***

- **Specify binary TOC or index for HTML Help.** Newly documented settings `BinaryTOC` and `BinaryIndex`; see §18.9.2 [Choosing whether to generate binary contents or index](#) on page 335.
- **Generate a CSH map file.** New settings `MakeCshMapFile`, `CshMapFileNumStart`, and `CshMapFileNumIncrement`; see §18.11.3 [Specifying and generating a map file for CSH links](#) on page 342.

#### ***Generating OmniHelp:***

- **Make compound terms searchable in OmniHelp.** Newly documented setting `CompoundWordChars`; see §19.10.3 [Making compound terms searchable](#) on page 372.

#### ***Converting to DITA XML:***

- **Wrap anchored images in `<fig>` as an exception.** Newly documented HTML format property `Figure`; see §24.7.2 [Specifying what to include in a `<fig>` wrapper](#) on page 483.
- **Keep selected topics out of the TOC.** New PI marker type `DITANoTOC`; see §24.8.6 [Omitting a DITA topic from the TOC](#) on page 489.
- **CSH targets via `TopicAlias` markers are included by default.** New setting `UseTopicAlias`; see §24.10 [Including CSH targets in DITA XML](#) on page 491.

#### ***Creating HTML links:***

- **Tell DITA2Go 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 **DITA2Go** to leave case alone in hypertext links, you must explicitly set `MakeFileHrefsLower` to `No` in a project or local configuration file; see §28.2.6 [Forcing link text to lowercase](#) on page 549.

#### ***Mapping text formats to HTML/XML:***

- **Assign HTML tags in new configuration sections.** `[ParaTags]` and `[CharTags]` replace `[ParaStyles]` and `[CharStyles]`, respectively; see:
  - §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566
  - §30.3 [Mapping character formats](#) on page 569
- **Distinguish paragraph (block) from character (inline) format properties.** New sections `[HTMLParaStyles]` and `[HTMLCharStyles]` supersede `[HTMLStyles]`, which is still honored; see §30.4 [Assigning properties to text formats](#) on page 570.
- **Exclude size attribute from font tags.** Newly documented setting `UseFontSize`; see §30.6.4 [Excluding face and size attributes from font tags](#) on page 578.

**Including graphics in HTML:**

- **Indent images without using CSS.** Only if CSS is not an option; see §32.5.2 [Indenting images \(deprecated\)](#) on page 617.

**Converting tables to HTML:**

- **Indent tables without using CSS.** Only if CSS is not an option; see §33.5.1 [Indenting tables \(deprecated\)](#) on page 641.

**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 §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

**Working with processing instructions:**

- **Replace PI marker content with code.** New section [MarkerTypeCodeReplace]; see §38.5 [Inserting code with PI markers](#) on page 724.

**August 21, 2012 update 03****Generating Microsoft HTML Help:**

- **Exclude selected topics from FTS for HTML Help.** An inelegant hack via @search or new PI marker **Search**; see §18.10 [Providing full-text search \(FTS\) for HTML Help](#) on page 339.

**Converting to HTML/XHTML:**

- **Prevent adjacent HTML <pre> elements from merging.** New setting **MergePre**; see §22.6.4 [Preventing adjacent <pre> elements from merging](#) on page 439.

**Setting up CSS for HTML:**

- **Omit element tags from CSS selectors.** New setting **SelectorIncludesTag**; see §31.7.9 [Omitting tags from CSS selectors](#) on page 605.

**Converting tables to HTML:**

- **HTML table rows are now wrapped in groups by default.** Reversed setting **HeadFootBodyTags**; see §33.3.2.4 [Wrapping table row groups](#) on page 629.

**May 1, 2012, version 4.0, update 54****Editing configuration files:**

- **Override, or produce the effect of, the @print attribute.** New setting **PrintProject**, new PI marker **Print**; see §4.1.7 [Producing print output selectively](#) on page 71.
- **Pop up current log file if conversion errors occur.** New settings **EditorFileName** and **HistoryFileName**; see §4.2 [Logging conversion events](#) on page 74.

**Mapping elements to output formats:**

- **Ignore outputclass attributes for border and shading properties.** New setting **OutputclassHasBorderShadeFormats**; see §6.2 [Specifying options for naming formats](#) on page 87.

- **Specify a default heading for <note> elements.** The **DITA2Go** default is no heading; see §6.5.8 [Understanding default attribute-based prefixes and headings](#) on page 100.
- **Designate a table row as a table footer row on output.** New setting **TableFooterClass**; see §6.9 [Specifying formats and options for tables](#) on page 103.
- **Give draft comments special formats;** see §6.11 [Specifying options for draft comments](#) on page 105.

#### **Configuring format components:**

- **Include page numbers, text, and spaces in cross-reference formats.** Previously undocumented building blocks; see [Table 8-5](#) on page 155.

#### **Specifying conditional processing:**

- **Limit the scope of keydefs to named map branches.** New settings **UseBranchKeydefs** and **KeydefsOnlyWithinBranch**; see §9.6.6 [Limiting the scope of keydefs by branch](#) on page 172.
- **Keyref named map branches.** New PI marker type **KeyrefBranch**; see §9.6.7 [Directing a key reference to the correct branch](#) on page 173.

#### **Including content by reference:**

- **Push to more places than specified for DITA 1.2.** New push actions **pushatstart** and **pushatend**; see §10.1 [Pushing and pulling content by reference](#) on page 175.
- **Include external code snippets.** New PI marker types **ExtCode\***, plus support for RFC 5147 fragment identifiers; see §10.2 [Referencing external code or text fragments](#) on page 176.

#### **Defining element sets and properties:**

- **Assign element type properties to class attributes.** Not just to element names; see:
  - §11.2.3 [Assigning properties to element types](#) on page 183
  - §B [Element type default properties](#) on page 825.
- **New element type properties:** **CascadeSet**, **CascadeItem**, **Draft**, **Abbrev**, **NoLevel**, **NoNumber**, **Task**, **BookTitle**, **PList**, **PLEntry**, **PLTerm**, **PLDef**, **Navtitle**, **Reference**, **Glossary**, and **Trademark**; see:
  - §11.2.2 [Understanding what properties are available](#) on page 180
  - §B [Element type default properties](#) on page 825

#### **Generating lists and indexes:**

- **Specify different suffixes for the base names of generated files.** New settings **TOCSuffix**, **LOFSuffix**, **LOTSuffix**, and **IDXSuffix**; see §14.2 [Naming generated HTML list and index files](#) on page 198.
- **Get navigation titles into the TOC.** And set defaults to use if attributes are missing. New settings **TopicheadsHaveNavtitles**, **LockAllNavtitles**, **UseAllInTOC**; see §14.3 [Generating a table of contents](#) on page 198.

#### **Converting to print RTF:**

- **Produce .doc or .docx files via Word macro.** See §15.10.1 [Supporting more than one version of Word](#) on page 239.

#### **Producing on-line Help:**

- **Choose plain or modified topic titles for TOC entries.** New setting **UseNavtitleMarkers**; see §16.4.2 [Including contents entries in HTML-based Help](#) on page 250.
- **Produce index meta elements for Microsoft Help Viewer.** New setting **UseHVIndex**; see §16.5.2 [Preparing index entries for Microsoft Help Viewer](#) on page 252.

**Generating Microsoft HTML Help:**

- **Omit code-page mapping for uncompiled HTML Help.** New setting `UseCodePage`; see §18.3.5 [Deciding whether to compile HTML Help](#) on page 317.
- **Generate Asian or Cyrillic HTML Help.** New code-page DLLs; see §18.12 [Generating HTML Help in non-Western languages](#) on page 344.
- **Fixed spaces become ideographs for Japanese HTML Help.** Or you can map them to something else; see §18.12 [Generating HTML Help in non-Western languages](#) on page 344.
- **Exclude content from full-text search.** If no `@search` attributes in your topicrefs, use new PI marker **Search**; see §19.10.7 [Excluding content from being searched](#) on page 374.

**Converting to HTML/XHTML:**

- **For ePub, XHTML output can provide input to Calibre.** See §22.1 [Deciding which type of output to produce](#) on page 430.
- **For HTML 5 output, set DOCTYPE as appropriate.** See §22.4.1 [Specifying HTML/XML version, DOCTYPE, and DTD](#) on page 432.
- **Generate XHTML for Confluence 4.x.** New settings `Confluence`, `ConfluenceLinks`, and `ConfluenceLink*`; see §22.10 [Generating XHTML for Confluence 4.x](#) on page 442.
- **Change the View Output starting topic for HTML/XHTML.** New setting `ViewOutputFile`; see §22.12 [Specifying a starting topic for HTML or XHTML](#) on page 443.

**Splitting and extracting files:**

- **Insert space or a separator between HTML topics in a single output file.** New `[Inserts]` keyword `TopicBreak`; see:
  - §27.6.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 535
  - §37.9.2 [Invoking macros at predetermined points in output](#) on page 710.

**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 §31.7.1 [Understanding CSS class name restrictions](#) on page 600.

**Including graphics in HTML:**

- **Omit empty alt attribute values from HTML output.** New setting `AllowEmptyAlt`; see §32.7 [Providing \(or omitting\) alternate text for images](#) on page 620.

**Converting tables to HTML:****Automating DITA2Go conversions:**

- **Execute system commands before and after conversion.** New automation keywords `SystemStartCommand`, `SystemWrapCommand`, `SystemEndCommand`; see §43.1 [Executing operating-system commands](#) on page 777.

**Producing deliverable results:**

- **Gather referenced graphics files for distribution.** New setting `CopyOriginalGraphics`; see §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796.

## Colophon

PDF edition

The PDF edition of the **DITA2Go User's Guide** is:

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

*HTML edition* The OmniHelp edition of the **DITA2Go 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

*Source files* The **DITA2Go User's Guide** is an unstructured FrameMaker version 8.0 document, converted to DITA XML using **Mif2Go**, and from DITA XML to all output formats (except PDF) using **DITA2Go**. The FrameMaker source files in `UGDITA2Go_frm4.zip` include path information and the configuration files used to generate DITA XML, as well as those used to generate all other outputs from DITA XML. You can download this archive from Omni Systems:

<http://www.dita2go.com>

The DITA XML source files, in `d2gug_demo.zip`, are also available for download. Log in and go to **Download > User's Guide**.

(5/19/13 13:14:00)



# 1 Getting started with DITA2Go

---

This section tells you how to install **DITA2Go**, and how to update **DITA2Go**. Topics include:

- §1.1 [What you need to know](#) on page 25
- §1.2 [What you need to have](#) on page 27
- §1.3 [What you need to do](#) on page 28
- §1.4 [How to update DITA2Go](#) on page 36
- §1.5 [How DITA2Go works](#) on page 37
- §1.6 [How to start and stop DITA2Go](#) on page 38
- §1.7 [How to work with DITA2Go](#) on page 38
- §1.8 [How to uninstall DITA2Go](#) on page 38

## 1.1 What you need to know

To use **DITA2Go** effectively, it is best if all the following apply:

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

*In this section:*

- §1.1.1 [How DITA2Go is organized](#) on page 25
- §1.1.2 [File, directory, and path names](#) on page 26
- §1.1.3 [Output types you can specify](#) on page 26
- §1.1.4 [Languages and character sets](#) on page 27

### 1.1.1 How DITA2Go is organized

**DITA2Go** is organized around the idea of *formats*, which are packages of presentational content, just as *elements* are packages of semantic content.

To convert your DITA source to an HTML or RTF representation, **DITA2Go** provides the means to perform two primary tasks:

- Map DITA elements to output formats.
- Define presentational properties of those output formats.

**DITA2Go** also carries out a number of output-type-dependent secondary tasks, such as constructing Help file infrastructure.

For each element in your DITA document, **DITA2Go** considers the element and its context, then maps the content to an appropriate output style. You can change the mapping, add exceptions, and revise the styles.

To map DITA elements to output formats, **DITA2Go** relies on both rules and instance mark-up. Rules come from settings in configuration files; instance mark-up is in the DITA files themselves, either as @outputclass values or as processing instructions (PIs).

To define format presentation, **DITA2Go** uses rules based on settings in format configuration files that include output formats for text and tables, and also for headers and footers and page layout formats for RTF output. Your **DITA2Go** distribution comes with a



set of predefined output formats. You can override the definitions of these formats, and define additional formats of your own. **DITA2Go** provides a way to specify presentation that works for both HTML and RTF output, which are radically different output types. The method is based mainly on CSS, but using configuration-file syntax, with extensions to support native CSS and RTF as needed. This approach incorporates the notion of “based” formats, as in Word, so that entire branches can be adjusted with just one setting.

For output-type-dependent transformations, the rules are in chains of output-type-specific configuration files; for a particular document, the rules are in DITA-source-specific configuration files. Configuration inheritance simplifies common styling across a set of projects, using a single-source set of formats.

### 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*

Files referenced by your DITA document, or by configuration settings, should have names and paths that conform to the following guidelines:

- Preferably no spaces in file or path names.
- 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.

*DITA2Go is not the problem!*

Although **DITA2Go** 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	PDF
WinHelp 4/95	Generic XML	JavaHelp	
	DITA XML	Oracle Help for Java	
	DocBook XML	OmniHelp	

**Print RTF**

**DITA2Go** handles styles, tables, and graphics. See §15 [Converting to print RTF](#) on page 219 for more information.

**WinHelp RTF**

You can configure and generate WinHelp RTF with **DITA2Go**; and if you have access to Help Workshop (unfortunately no longer available from Microsoft) you can compile WinHelp files that need no further tweaking. **DITA2Go** produces all the files you need, including a TOC, an Index, and a Help project file. See:

§17 [Generating WinHelp](#) on page 281.



<b>HTML-based Help</b>	<p>You can use <b>DITA2Go</b> to configure and generate several flavors of HTML-based Help. <b>DITA2Go</b> produces all the files you need, typically including TOC, Index, and a Help project file. For the two Java formats, <b>DITA2Go</b> prepares the map file; for HTML Help, the aliases file. See:</p> <ul style="list-style-type: none"> <li>§18 <a href="#">Generating Microsoft HTML Help</a> on page 313</li> <li>§19 <a href="#">Generating OmniHelp</a> on page 353</li> <li>§20 <a href="#">Generating JavaHelp or Oracle Help</a> on page 385</li> <li>§21 <a href="#">Generating Eclipse Help</a> on page 413.</li> </ul>
<b>HTML and XML</b>	<p><b>DITA2Go</b> can produce HTML 4.01, XHTML 1.0, and XML 1.0 files from your DITA document, and also create Cascading Style Sheets. You can include arbitrary JavaScript anywhere in HTML output. You can produce ePub input from <b>DITA2Go</b> XHTML output. See:</p> <ul style="list-style-type: none"> <li>§22 <a href="#">Converting to HTML/XHTML</a> on page 429</li> <li>§23 <a href="#">Converting to generic XML</a> on page 449.</li> </ul>
<b>DITA and DocBook</b>	<p><b>DITA2Go</b> can produce DITA XML and DocBook XML from DITA documents. See:</p> <ul style="list-style-type: none"> <li>§24 <a href="#">Converting to DITA XML</a> on page 455</li> <li>§26 <a href="#">Converting to DocBook XML</a> on page 499.</li> </ul>
<b>PDF</b>	<p><b>DITA2Go</b> produces PDF output indirectly, through a Word macro; once you have that in place, you can generate PDF output with one click. See:</p> <ul style="list-style-type: none"> <li>§15.1.5 <a href="#">Producing PDF automatically via Word</a> on page 222</li> </ul>
<b>Intermediate format</b>	<p>You can run a <b>DITA2Go</b> 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 <b>DITA2Go</b> again to continue the conversion.</p>

### 1.1.4 Languages and character sets

In addition to Western languages, **DITA2Go** supports Russian, Greek, and Central/Eastern European languages. For HTML/XML outputs **DITA2Go** supports *all* languages, via Unicode (UTF-8). For RTF outputs, **DITA2Go** supports only single-byte languages, although it is possible to produce decent Japanese RTF for Word.

**DITA2Go** does not currently support non-Unicode double-byte languages, nor right-to-left languages such as Hebrew and Arabic. However, if you use **DITA2Go** 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 27.

## 1.2 What you need to have

To use **DITA2Go** 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 **DITA2Go**, you will need at least some of the following software:

[XML editor](#)  
[Text editor](#)  
[File comparison tool \(optional\)](#)  
[Archiving tool](#)

### Ancillary tools for Help output.

<i>XML editor</i>	You can integrate <b>DITA2Go</b> with <oXygen/>, using external commands. See §1.3.11 <a href="#">Integrate DITA2Go with &lt;oXygen/&gt; (optional)</a> on page 36. However, you might find it more efficient to use the <b>DITA2Go</b> Project Manager; see §2.5 <a href="#">Running a DITA2Go conversion</a> on page 44.										
<i>Text editor</i>	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.										
<i>File comparison tool (optional)</i>	If you modify any of the local configuration files provided with your <b>DITA2Go</b> distribution, you might find it helpful to have a file comparison tool such as WinMerge; see §1.3.9 <a href="#">Obtain a file comparison tool (optional)</a> on page 35.										
<i>Archiving tool</i>	To have <b>DITA2Go</b> 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.										
<i>Ancillary tools for Help output</i>	If you intend to generate on-line Help, you will need one or more of the following tools; see §1.3.6 <a href="#">Obtain tools for Help systems or eBooks</a> on page 32: <table> <tr> <td>WinHelp</td><td>Microsoft Help Workshop (<i>no longer available</i>) and viewer</td></tr> <tr> <td>HTML Help</td><td>Microsoft HTML Help Workshop</td></tr> <tr> <td>JavaHelp</td><td>Java Runtime Environment (JRE) and JavaHelp software</td></tr> <tr> <td>Oracle Help for Java</td><td>Oracle Help for Java Developer's Kit 2.0</td></tr> <tr> <td>Eclipse Help</td><td>Java Runtime Environment (JRE), Eclipse Platform</td></tr> </table>	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
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

Follow the instructions in this section the first time you install a version of **DITA2Go**. To update **DITA2Go**, see §1.4 [How to update DITA2Go](#) on page 36.

*In this section:*

- §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29
- §1.3.2 [Download a DITA2Go distribution](#) on page 30
- §1.3.3 [Install DITA2Go](#) on page 30
- §1.3.4 [Make Omni Systems executables accessible](#) on page 31
- §1.3.5 [Check your DITA2Go installation](#) on page 32
- §1.3.6 [Obtain tools for Help systems or eBooks](#) on page 32
- §1.3.7 [Establish system-wide configuration settings](#) on page 33
- §1.3.8 [Locate document-specific settings](#) on page 35
- §1.3.9 [Obtain a file comparison tool \(optional\)](#) on page 35
- §1.3.10 [Download the DITA2Go User's Guide \(optional\)](#) on page 36
- §1.3.11 [Integrate DITA2Go with <oXygen/> \(optional\)](#) on page 36

### 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
- 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 §A.1.4 [Check path names, file names, and drive location](#) on page 820.

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:

`%omsyshome%\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:

`;%omsyshome%\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 **DITA2Go** will be able to find all the Omni Systems files.

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 DITA2Go distribution](#) on page 30.

## 1.3.2 Download a DITA2Go distribution

Register (or log in) at:

<http://www.dita2go.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:

**DITA2Go**, any version

**Mif2Go** version 4.0 or later, but not **DITA2Go**

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

### What to download:

d2g\_update\_4.zip

d2g\_addon\_4.zip

d2g\_full\_4.zip

Next: §1.3.3 [Install DITA2Go](#) on page 30.

## 1.3.3 Install DITA2Go

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 29.
- If you already have **DITA2Go** on your system, skip this section and instead §1.4.1 [Update your DITA2Go installation](#) on page 37.

*Why no installer?*

Omni Systems does not provide an installer for **DITA2Go**. This is for transparency; you *know* **DITA2Go** 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 DITA2Go distribution archive](#) on page 30

§1.3.3.2 [Finish installing DITA2Go](#) on page 31

### 1.3.3.1 Extract files from the DITA2Go distribution archive

To install **DITA2Go** for the first time, place **DITA2Go** distribution d2g\_full\_4.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 29).

For example, executables should be here:

%OMSYSHOME%\common\bin

*not here:*

%OMSYSHOME%\d2g\_full\_4\common\bin

*Check directory structure*

You should have a directory structure that looks like this:

```
%OMSYSHOME%
|
|--common
|   |--bin
|   |--local
|   +--system
```

```

|--d2g
|   |--documents
|   |--dtds
|   |--local
|   |--specializations
|   |--system
|   |--usersguide
|   +--zip
+--demo
    +--DITATestSuite

```

### 1.3.3.2 Finish installing DITA2Go

To complete the installation:

1. Move the **DITA2Go** distribution .zip file to subdirectory %OMSYSHOME%\d2g\zip, where it will be available for future reference.
2. On your desktop, create shortcuts to:  
 %OMSYSHOME%\common\bin\d2gpm.exe  
 %OMSYSHOME%\common\bin\d2gcm.exe

This gives you double-click access to the following tools:

**DITA2Go Project Manager:** create, modify, and run conversion projects

**DITA2Go Configuration Manager:** edit project settings.

See §2 [Converting DITA documents](#) on page 39.

3. In Windows Explorer, navigate to:  
 %OMSYSHOME%\d2g\usersguide

Right-click ugdita2go.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 ugdita2go.chm to register the **DITA2Go** context-sensitive Help system with Windows, so **DITA2Go** can find it.
5. On your desktop, create a shortcut to:  
 %OMSYSHOME%\d2g\usersguide\ugdita2go.chm

This gives you access to the **DITA2Go User’s Guide**, HTML Help edition. You can download other editions from Omni Systems; see [Availability](#) on page 17.

Next: §1.3.4 [Make Omni Systems executables accessible](#) on page 31.

### 1.3.4 Make Omni Systems executables accessible

Starting with **DITA2Go** version 4.0, all Omni Systems executables 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 29.

*Old Mif2Go files*

If you have **Mif2Go** version 3.3 installed on your system (even the evaluation version), do the following:

1. Move drmif.dll from your Windows system directory to your new Omni Systems executables directory, %omsyshome%\common\bin (see § [Create an Omni Systems environment variable](#) on page 29).

2. 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 that work for both **Mif2Go** and **DITA2Go**:

```
dcl.exe
dwrftf.dll
dwhtm.dll
dwinf.dll
dwinf.dll
libexpat.dll
```

**Note:** If you leave the old files in the system directory, Windows will use them instead of your new **DITA2Go** executables, and you will wonder why **DITA2Go** does not work as expected.

*Automated build systems*

If you have been using automated build systems for **DITA2Go** that rely on finding executables in the Windows system directory, you have two choices:

- Change your scripts to access %omsyshome%\common\bin instead; *this is the preferred method.*
- Every time you update **DITA2Go**, copy all the new executables from %omsyshome%\common\bin to the Windows system directory.

Next, §1.3.5 [Check your DITA2Go installation](#) on page 32.

### 1.3.5 Check your DITA2Go installation

To make sure everything is set up correctly, try running the DITA Test Suite demonstration project included with the **DITA2Go** distribution:

1. Start the **DITA2Go** Project Manager (if you do not already have a desktop shortcut to this program, see [Step 2](#) under §1.3.3 [Install DITA2Go](#) on page 30).
2. On the **Run Project** tab, select project DTS Word Demo (for RTF output) or DTS HTML Demo (for HTML output).
3. Click **Start**.

If the project produces output and a log file, your installation is fine. If the project produces an error message, or appears to run instantly without producing a log file, this means the Project Manager did not find the necessary files.

Next, if you plan to produce on-line Help or ebook formats, §1.3.6 [Obtain tools for Help systems or eBooks](#) on page 32; otherwise, §1.3.7 [Establish system-wide configuration settings](#) on page 33.

### 1.3.6 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 §18.12 [Generating HTML Help in non-Western languages](#) on page 344.

*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 §17.1 [Obtaining tools for WinHelp](#) on page 281.
- eBooks* You can produce ePub input with **DITA2Go** XHTML output, then use free converter Calibre: <http://calibre-ebook.com>

Next, §1.3.7 [Establish system-wide configuration settings](#) on page 33.

### 1.3.7 Establish system-wide configuration settings

Telling **DITA2Go** how to use certain tools will allow you to run conversions directly from the **DITA2Go** Project Manager.

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

```
%omsyshome%\common\local\config\local_omsys.ini
```

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

**Note:** For proper syntax, see §3.4 [Understanding the rules for configuration settings](#) on page 62.

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

- [XML catalogs](#)
- [Specialized DTD](#)
- [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](#)
- [View-output command](#)

*XML catalogs* If you have specialized, you must prepare an XML catalog for your specialized DTDs, list their local paths here, and define an access key for each. See §4.1.1 [Connecting to XML catalogs](#) on page 67.

*Specialized DTD* By default, **DITA2Go** uses the Oasis DITA 1.1 DTD located in directory %OMSYSHOME%\common\dtDs\dita1.1, or whichever catalog you have specified. If your DITA XML includes specializations for which you did not create a catalog, you must tell **DITA2Go** where to find your own DTD:

```
[Setup]
DTDPath=path\to\specialized\dtD
```

See §4.1.3 [Specifying a DITA XML DTD](#) on page 68.

*Archiving program and options* **DITA2Go** 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 §44.11 [Archiving deliverables](#) on page 803. You can use the **DITA2Go** Project Manager to specify an archive name and version for each project. 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, **DITA2Go** 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 **DITA2Go** where to find it:

```

[MSHtmlHelpOptions]
Compiler = path\to\hhc

```

See §18.13.1 [Directing DITA2Go to run the HTML Help compiler](#) on page 346.

*Eclipse Help zip  
command*

For Eclipse Help projects, **DITA2Go** 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 §21.8 [Packaging Eclipse Help files](#) on page 427.

*JavaHelp, Oracle  
Help index and  
JAR commands*

For JavaHelp projects, **DITA2Go** 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, **DITA2Go** can run the Oracle Help indexer to produce a full-text search index:

```

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

```

See §20.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 397, and §20.6.1 [Creating a JAR file](#) on page 400.

*WinHelp  
copyright  
statement and  
compiler  
command*

For WinHelp projects, **DITA2Go** can provide a copyright statement to be included in the WinHelp .hlp project file:

```

[HelpOptions]
HelpCopyright = your copyright statement

```

**DITA2Go** 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 **DITA2Go** where to find it. For example:

```

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

```

See §17.2.10 [Compiling a WinHelp project](#) on page 285.

*View-output  
command*

When you use the **DITA2Go** Project Manager to run a conversion, you can view the output immediately with the **View Output** button on the **Run Project** tab, provided the Project Manager can launch an appropriate viewer:



- For HTML, XHTML, and generic XML output, no command is needed; the TOC file (if any) opens in the default browser. See §22.12 [Specifying a starting topic for HTML or XHTML](#) on page 443.
- For HTML Help and WinHelp, no command is needed to run the compiled Help system in the wrap directory; for OmniHelp, the default topic opens in the default browser.
- For all other outputs, you must provide a Windows command the Project Manager can execute:

```
[*Options]
; ViewOutputCommand = path\to\viewer, default none
```

You can specify an absolute path or a path relative to the wrap directory. For example.

```
[JavaHelpOptions]
ViewOutputCommand = java -jar D:\JH2\demos\bin\hsvviewer.jar -helpset
```

Specify each view-output command in an options configuration section specific to the output type:

<u>Output type</u>	<u>Options configuration section</u>
DITA XML	[DITAOptions]
Docbook XML	[DocBookOptions]
Eclipse Help	[EclipseHelpOptions]
HTML/XHTML	[HTMLOptions]
JavaHelp	[JavaHelpOptions]
Microsoft HTML Help	[MSHtmlHelpOptions]
Microsoft Word	[WordOptions]
OmniHelp	[OmniHelpOptions]
Oracle Help for Java	[OracleHelpOptions]
WinHelp	[HelpOptions]
XML (flat)	[HTMLOptions]

### 1.3.8 Locate document-specific settings

You can choose a default location for document-specific configuration files on the **Preferences** tab of the **DITA2Go** Project Manager; see §2.6 [Customizing the DITA2Go Project Manager](#) on page 45. To determine which location is appropriate, see §39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733.

Your **DITA2Go** installation is now complete, and you can set up a **DITA2Go** conversion project, or run an existing conversion. Your existing project configuration files will work without modification. See §2 [Converting DITA documents](#) on page 39.

### 1.3.9 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 **DITA2Go** distributions, with WinMerge you will be able to see what has changed when you update the system copies.

### 1.3.10 Download the DITA2Go User's Guide (optional)

The HTML Help edition of the **DITA2Go User's Guide** is included with your **DITA2Go** installation. You can download other editions of the **DITA2Go User's Guide**, in several formats, from the Omni Systems Web site:

<http://www.dita2go.com/>

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

### 1.3.11 Integrate DITA2Go with <oXygen/> (optional)

You can integrate **DITA2Go** with XML editor <oXygen/>, using external commands. You will need to set up a command for each output format you wish to produce,

1. Use the **DITA2Go** Project Manager to set up a starting project configuration file in an appropriate output directory; see §2.1 [Creating a DITA2Go conversion project](#) on page 39. Use the recommended directory structure, with the output directory under the directory where your map is located.
2. Start <oXygen/>.
3. Go to **Tools > External Tools > Preferences**; select **Global Options**; Click **New**.
4. Enter any name and description, leaving encoding at defaults; pick any shortcut key.
5. For HTML output (for example), set **Working Directory** to:

```
{cfd}\html
```

6. Set **Command Line** to:

```
dcl -f html ..\${cfne}
```

7. Click **OK**.

For other output formats, change the **Working Directory** name and the **-f** option from `html` to another format, such as `rtf` or `omnihelp`; see §2.7.2 [Understanding how to run DITA2Go DCL](#) on page 47.

To publish or transform your topic or map, select the map in the Editor, make sure the map is the active window, then use **Tools > External Tools > YourCommand**, and go. Or just use the toolbar button <oXygen/> creates for your command.

This procedure assumes you are using the recommended directory structure, with the output directory under the directory where your map is located. If you have a different directory structure, you will need to make some adjustments; see §2.7 [Converting documents from the command line](#) on page 46.

## 1.4 How to update DITA2Go

If you have never installed **DITA2Go** on your system, you must first §1.3.3 [Install DITA2Go](#) on page 30. Then in future you can use one of the update procedures described here.

*In this section:*

§1.4.1 [Update your DITA2Go installation](#) on page 37

§1.4.2 [Try out DITA2Go beta executables](#) on page 37

### 1.4.1 Update your DITA2Go installation

As long as you subscribe to **DITA2Go** support, you are automatically notified of new versions of **DITA2Go**.

**Note:** Updates do not change anything in your %OMSYSHOME%\d2g\local or %OMSYSHOME%\common\local subdirectories, nor replace executables that are unchanged since the prior update.

To update **DITA2Go** to the latest version:

1. Download d2g\_update\_4.zip into your Omni Systems home directory, %OMSYSHOME%.
2. Extract all files from d2g\_update\_4.zip, allowing old files to be overwritten.

### 1.4.2 Try out DITA2Go beta executables

You can obtain the latest beta revisions of **DITA2Go** distribution files dr\*.dll and dw\*.dll from the Omni Systems Web site:

<http://www.dita2go.com/>

Log in, go to **Download > Basic 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 dwrtf372.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.

## 1.5 How DITA2Go works

**DITA2Go** works on map files, not directly on .dita files. If necessary, you can use auxiliary program **DITA2Map** to generate a map file from your DITA files; see §46 [Creating a map with DITA2Map](#) on page 815.

**DITA2Go** can work with files anywhere on the local system or on mapped drives (in Windows). **DITA2Go** does not attempt to duplicate the source directory structure in the output, but instead creates by default a flat structure, and by adjusting the HTML names, ensures that there are no name conflicts.

If you really need a directory structure for output, you can have it, but the structure cannot go above the level of the default output directory. This is a limitation imposed by many Help formats, such as CHM for HTML Help. **DITA2Go** handles such output structure requirements so they do not dictate the input structure you can use.

With the **DITA2Go** Project Manager, a simple tabbed dialog, you can set up **DITA2Go** projects, edit your project configuration files, run conversions, and inspect the resulting log files. When you click **Start** on the **Run Project** tab, Document Coding Language filter dcl.exe is invoked; see §45 [Converting via DCL](#) on page 809.

First, the files in your DITA document are converted to an intermediate representation via input filter `drxml.dll`.

Next, the intermediate representation is converted to whichever output format your starting configuration file specifies, via output filter `dwhrm.dll` (for HTML/XML outputs) or `dwrtrf.dll` (for Word or WinHelp outputs).

## 1.6 How to start and stop DITA2Go

**Starting DITA2Go** On your desktop, double-click the shortcut to the **DITA2Go** Project Manager. (If you do not already have a desktop shortcut to this program, see [Step 2](#) under §1.3.3 **Install DITA2Go** on page 30).

**Stopping DITA2Go** If you are running a conversion from the **DITA2Go** Project Manager, on the **Run** tab click **Stop**. If the Windows command window is open, you will have to close it manually.

If you are running a conversion from the command line, how to cancel a **DITA2Go** conversion before it finishes depends on which version of Windows you are using. Try one of the following:

- Press **Ctrl+C**, which works in some Windows environments.
- Open the Windows Task Manager (on Windows 2000/XP/Vista/7):
  1. Select the **DITA2Go** process, `dc1.exe`.
  2. Click **End Task**.

## 1.7 How to work with DITA2Go

The methodology for converting documents with **DITA2Go** 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 **DITA2Go 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 DITA2Go

To uninstall **DITA2Go**, delete all files and subdirectories from directory `%OMSYSHOME%\d2g`. If you have no other Omni Systems applications installed, you can also delete the Omni Systems home directory and all subdirectories.

# 2 Converting DITA documents

---

This section shows how to set up a conversion project and use **DITA2Go** to convert DITA XML documents to another format. Topics include:

- §2.1 [Creating a DITA2Go conversion project](#) on page 39
- §2.2 [Modifying a DITA2Go conversion project](#) on page 40
- §2.3 [Configuring default DITA2Go project settings](#) on page 40
- §2.4 [Inspecting and editing configuration files](#) on page 44
- §2.5 [Running a DITA2Go conversion](#) on page 44
- §2.6 [Customizing the DITA2Go Project Manager](#) on page 45
- §2.7 [Converting documents from the command line](#) on page 46

*See also:*

- §5 [Modifying output appearance](#) on page 79

## 2.1 Creating a DITA2Go conversion project

To create a **DITA2Go** project, start the **DITA2Go** Project Manager. (If you do not already have a desktop shortcut to this program, see [Finish installing DITA2Go, Step 2](#) under §1.3.3 [Install DITA2Go](#) on page 30).

On the **Create** tab:

1. For **Project name**, specify a name for your conversion project; or, to start with the same configuration as an existing project, select the name of that project and change it to a new name.
2. For **DITA source directory**, browse to the directory where the DITA map file you wish to use for this conversion is located. It does not matter where the files referenced in the map are located, as long as those files are accessible.
3. For **DITA source map file**, browse to the bookmap or to a ditamap to use for the project. If your DITA document does not include any map files, select a topic file as a starting point, and **DITA2Go** will create a map for the document.

**Note:** If there is a bookmap in the directory, **DITA2Go** will select it automatically, and you will not have to browse for it. If there is no bookmap, **DITA2Go** will check for a ditamap; if no ditamap, for a `.dita` file. If none of those are present, **DITA2Go** will show the last map used in an active project.

4. For **Output type**, select the output format you want **DITA2Go** to produce. The default output type is OmniHelp.
5. For **Output Directory**, use the *Browse* dialog to create a new directory for the output files to be produced with this project. A good place to create such a directory is under the directory where your DITA `.bookmap`, `.ditamap`, or `.dita` files are located.

**Note:** Never create more than one output in the same directory; they will conflict badly.

6. Click **Create Project**. The **DITA2Go** Project Manager copies a starting configuration file for the output type you specified, from your **DITA2Go** `d2g\local\starts` directory (or from `d2g\system\starts`) to the output directory; then **DITA2Go** saves the project options you specified.

7. Click **Configure Defaults** to use a wizard to establish values for basic project settings; see §2.3 [Configuring default DITA2Go project settings](#) on page 40.
8. Click **Edit Ini Files** to edit the starting and source-specific configuration files directly; see § [Specify a text editor](#) on page 45.

## 2.2 Modifying a DITA2Go conversion project

You can use the **DITA2Go** Project Manager to change the map file, the source directory, or the output directory for an existing **DITA2Go** conversion project.

On the **Modify Project** tab:

1. Select the project to modify.
2. Change to a different DITA source directory *only if you have moved the DITA document files for this project*.
3. Select a map file: bookmap or ditamap. The default is the map file specified when the project was created.
4. Change the output type *only to correct a mistaken choice*; otherwise, create a new project instead.
5. Optionally, browse to, or even create, a different output directory.
6. Click **Modify Project**. The **DITA2Go** Project Manager saves the options you changed, in %OMSYSHOME%\d2g\system\projects.ini.
7. Click **Configure Defaults** to use a wizard to change additional settings; see §2.3 [Configuring default DITA2Go project settings](#) on page 40. Or click **Edit Ini Files** to edit the starting and source-specific configuration files directly; see § [Specify a text editor](#) on page 45.

## 2.3 Configuring default DITA2Go project settings

Once you have created or modified a conversion project, you can use the **DITA2Go** Project Manager to establish or alter additional project settings. When you click **Configure Defaults**, the Configuration Wizard guides you through a series of steps to specify default values for certain settings.

*In this section:*

- §2.3.1 [Understanding where to specify configuration settings](#) on page 40
- §2.3.2 [Choosing a source-specific configuration file](#) on page 41
- §2.3.3 [Deciding whether to compile and assemble output](#) on page 41
- §2.3.4 [Specifying a ditaval file](#) on page 41
- §2.3.5 [Naming an archive for output](#) on page 42
- §2.3.6 [Assembling graphics to include with output](#) on page 42
- §2.3.7 [Including output-specific settings](#) on page 42
- §2.3.8 [Reviewing initial project settings](#) on page 43

### 2.3.1 Understanding where to specify configuration settings

Some settings can be specified in either of the following configuration files:

*Source-specific:* Settings that apply to all or most conversions from the DITA source document you selected. This is the primary configuration file for all projects related to that DITA source document.

*Project:* Settings that apply only to the current conversion project, with values that override values of the same settings in the source-specific configuration file.

Other settings are specific to the output type you selected; those settings always go in the starting configuration file.

When you create the *first DITA2Go* conversion project for a given DITA source document, pretty much every setting (except those specific to the output type) should go in the source-specific configuration file with a reasonable value that serves as the default for all outputs from that DITA source. Then you can override that value as needed for individual conversion projects.

When you create *another DITA2Go* conversion project for the same DITA source document, changing or negating a setting in the source-specific configuration file might not be a wise move:

- If other projects have that same setting in their starting configuration files, a value you change in the source-specific configuration file will be ignored for those projects, because settings in a starting configuration file override settings in a source-specific configuration file. If you intend the new value to affect all such projects, you will have to reconfigure each of those projects.
- If other projects do not include that setting in their starting configuration files, those projects are relying on the value in the source-specific configuration file, which might no longer be appropriate. If you do not want the new value to affect all such projects, you will have to reconfigure any for which it is inappropriate.

### 2.3.2 Choosing a source-specific configuration file

On the **Configure Default Settings** page, the Configuration Wizard displays the source-specific configuration file selected when you created or modified the current project. This will almost always be the file you want to use. However, if you have additional source-specific configuration files for the same source document, you can browse to one of those files instead, to use for the current project. Default settings that relate to the DITA source document will be placed in this file.

### 2.3.3 Deciding whether to compile and assemble output

On the **Configure Default Settings** page, you can choose whether to have **DITA2Go** compile Help (if this is a compilable Help project), and assemble all output components, including graphics, into a ready-to-ship deliverable. Unlike other settings that can go in either the source-specific or the starting configuration file, this particular setting is placed in both files. You can always override the value of this setting at run time; see §2.5 [Running a DITA2Go conversion](#) on page 44.

### 2.3.4 Specifying a ditaval file

On the **Conditional Options** page, the Configuration Wizard displays the last ditaval file selected, or just the source directory if no ditaval file was previously selected. You can do any of the following:

- If the appropriate ditaval file is already displayed, do nothing.
- Click **Use** and browse to a ditaval file to use for the current project.
- Click **None** if the current project does not need a ditaval file.

Next you can choose whether to store this ditaval setting in the source-specific configuration file, where it can be used for all conversions from your DITA source



document, or in the starting configuration file, where it will be used only for the current project.

**Note:** If other conversion projects already exist for this particular DITA document, you might not want to change or negate a ditaval entry in the source-specific configuration file; see §2.3.1 [Understanding where to specify configuration settings](#) on page 40.

### 2.3.5 Naming an archive for output

On the **Archive Name and Version** page, you can name an archive file for the output from a conversion. The base name of the archive file is a concatenation of the archive name and the archive version: *NameVersion.zip*.

The default archive name is the name of the map file selected for the current project; the default archive version is the name of the output type for the current project. For example, the default name for a conversion from the Omni Systems DITA test suite to WinHelp would be `DITATestSuitewinhelp.zip`. If you expect to produce future updates of the same project, you might want to use the project name for the archive name, and an incremental version number for the archive version; for example, `DTS2WH_01.zip`.

Use only letters, digits, and underscores in the archive name and version settings.

Most likely you will want to apply these settings to the current project only, rather than to all outputs from the current DITA source document.

### 2.3.6 Assembling graphics to include with output

On the **Graphics Assembly** page, you can specify graphics files to be included with output, and tell **DITA2Go** where to get those graphics.

**DITA2Go** automatically copies graphics files to the assembly directory for your project, from a single other directory. The default is the DITA source directory selected for the current project. You can browse to another directory.

For HTML and XML output types, by default **DITA2Go** copies all JPEG and GIF files from the directory you select. For RTF output types, by default **DITA2Go** copies all BMP and WMF files. You can specify other graphics types, and individual graphics file names, separated by spaces. You can include ? and \* wildcards in file specifications.

### 2.3.7 Including output-specific settings

For each output type, the Configuration Wizard includes a page of settings that apply only to the output type specified for the current project. All such settings go into the starting configuration file for the project. [Table 2-1](#) lists the output-specific settings available through the Configuration Wizard.

**Table 2-1 Configuration Wizard output-specific settings**

Output type	Configuration section	Keyword	Description	Ref.
OmniHelp	[HTMLOptions]	Title	Default page title	<a href="#">22.4.5</a>
	[OmniHelpOptions]	ProjectName	Help project name	<a href="#">19.3.2</a>
		HelpFileTitle	Help system title	<a href="#">19.3.4</a>
		DefaultTopicFile	Opening topic file	<a href="#">19.3.5</a>
	[OHTopLeftNav]	<i>Any code or macro for branding</i>		<a href="#">19.5.2</a>



**Table 2-1 Configuration Wizard output-specific settings (continued)**

Output type	Configuration section	Keyword	Description	Ref.
HTML Help	[HTMLOptions]	Title	Default page title	<a href="#">22.4.5</a>
	[MSHtmlHelpOptions]	HHPFileName	Help project file name	<a href="#">18.3.6</a>
		HelpFileName	Help system title	<a href="#">18.3.4</a>
		DefaultTopicFile	Opening topic file	<a href="#">18.3.7</a>
HTML 4.0	[HTMLOptions]	Title	Default page title	<a href="#">22.4.5</a>
XHTML 1.0	[HTMLOptions]	Title	Default page title	<a href="#">22.4.5</a>
Eclipse Help	[EclipseHelpOptions]	TocLabel	Help system title	<a href="#">21.4.3.1</a>
		PluginName	Name attribute	<a href="#">21.3.2</a>
		PluginID	ID attribute	<a href="#">21.3.2</a>
		PluginProvide	Provider attribute	<a href="#">21.3.2</a>
		TocTopic	Opening topic file	<a href="#">21.4.3.2</a>
JavaHelp	[HTMLOptions]	Title	Default page title	<a href="#">22.4.5</a>
	[JavaHelpOptions]	HSFileName	HelpSet file name	<a href="#">20.3.7.1</a>
		HelpFileName	Help system title	<a href="#">20.3.7.1</a>
		DefaultTopicFile	Opening topic ID	<a href="#">20.3.7.2</a>
Oracle Help	[HTMLOptions]	Title	Default page title	<a href="#">22.4.5</a>
	[OracleHelpOptions]	HSFileName	HelpSet file name	<a href="#">20.3.7.1</a>
		HelpFileName	Help system title	<a href="#">20.3.7.1</a>
		DefaultTopicFile	Opening topic ID	<a href="#">20.3.7.2</a>
Word	[FileIDs]	<i>Map file name</i>	Bookmark prefix	<a href="#">4.3.3</a>
	[WordOptions]	Word8	Version of MS Word	<a href="#">15.2</a>
		Word2000		
		Word2002		
		Word2003		
WinHelp	[HelpOptions]	HPJFileName	Help project file name	<a href="#">17.2.6</a>
	[HelpContents]	CntTitle	Contents title	<a href="#">17.12.1</a>
		CntTopic	Opening topic	<a href="#">17.12.1</a>

## 2.3.8 Reviewing initial project settings

On the **Review Settings** page, you can look over the settings you specified, presented as they will appear in both the source-specific and the starting configuration files:

- to make changes, click **Back**
- to have **DITA2Go** write these settings to their respective files, click **Finish**
- to discard all settings, click **Cancel**.

Depending on the type of output, the **DITA2Go** Project Manager adds to your starting configuration file certain settings that are not normally needed by the main program:

```
[Automation]
; MakeFTS = Yes (default, create a search index for JavaHelp or
; Oracle Help for Java) or No
MakeFTS = Yes
; MakeJar = Yes (default, package JavaHelp output in a .jar file) or No
MakeJar = Yes
; MakeArchive = Yes (default, archive output in a .zip file) or No
MakeArchive = Yes
; UseDCLSource = No (default) or Yes, convert from .dcl if available
UseDCLSource = No
```

```

; UseDCLOutput = No (default) or Yes, convert to .dcl instead of
; formatted output
UseDCLOutput = No

```

Based on the choices you make on the **Run Options** tab (see §2.5 [Running a DITA2Go conversion](#) on page 44), the Project Manager switches these settings on and off when appropriate for the output type you select.

## 2.4 Inspecting and editing configuration files

Once you have created or modified a conversion project, you can use the **DITA2Go** Project Manager to edit settings in the source-specific and the starting configuration files, and the **DITA2Go** Configuration Manager to inspect and edit all settings in all the configuration files you project accesses.

To inspect and edit source-specific and starting configuration files, click **Edit Ini Files**. The Project Manager opens these files for editing in the text editor of your choice; see § [Specify a text editor](#) on page 45.

To inspect and edit all settings in all configuration files that your project accesses, click **Configure**. The Configuration Manager opens with your project selected; see §3.1 [Working with DITA2Go configuration files](#) on page 49.

## 2.5 Running a DITA2Go conversion

Once you have created a conversion project with the **DITA2Go** Project Manager (see §2.1 [Creating a DITA2Go conversion project](#) on page 39), you can use the **DITA2Go** Project Manager to execute the conversion. However, you can also run **DITA2Go** from a command line; see §2.7 [Converting documents from the command line](#) on page 46.

To operate **DITA2Go** with the Project Manager:

1. On the **Run Project** tab, select the conversion project you wish to run. The **DITA2Go** Project Manager shows the DITA source directory, map file, and output directory, and the output type. You can browse to a different map file, perhaps to convert just a subset of your DITA document. The Project Manager also sets run options based on current settings in the starting project configuration file.
2. On the **Run Options** tab:

Under **Automation Options**, specify any post-conversion steps you need for the current run, or turn off those you do not need. With appropriate settings in place (see §1.3.7 [Establish system-wide configuration settings](#) on page 33), **DITA2Go** can produce a ready-to-distribute archive of your completed conversion, and place the archive in the shipping directory.

Under **Advanced Options**, you can choose whether to involve creation or use of an intermediate format: ASCII DCL. Your choice has the following effect:

<b>Source</b>	ASCII DCL files, if any, are converted to the output format.
<b>Output</b>	DITA XML files are converted to ASCII DCL.
<b>Neither</b>	DITA XML files are converted straight to the output format.
<b>Both</b>	DITA XML files are converted first to ASCII DCL, then the ASCII DCL files are converted to the output format.

When you run normally, with *Use ASCII DCL* set to **Both** or **Neither**, you get the full automation effect: both Start and End automation tasks. If instead you run with *Use*

*ASCII DCL* set to **Output**, you get only the Start automation tasks, because the files processed by End do not exist yet. If *Use ASCII DCL* is set to **Source**, you get only the End automation tasks, because the files removed by Start are needed for your current process.

Also, you can choose to run only automation tasks.

**Note:** If you choose *Run only automation tasks*, you must have the bookmap or ditamap in place in the output directory, even though it is not needed for these tasks.

3. On the **Run Project** tab, click **Start**. When the conversion is complete, you will see a message to that effect in the **Results:** field. If you chose run option **Wrap and Ship**, for some output types the **View Output** button is enabled by default. For other output types you will have to provide a view-output command to enable the button; see §1.3.7 [Establish system-wide configuration settings](#) on page 33.
4. Click **Show Log File** to see an event log that shows what happened during conversion.
5. Click **View Output** (if this button is enabled) to see the conversion output. To tell **DITA2Go** which file to open first on **View Output**, see §22.12 [Specifying a starting topic for HTML or XHTML](#) on page 443. The **View Output** button works nicely for XHTML, HTML, OmniHelp, HTML Help, and Word RTF output. However, for JavaHelp, Oracle Help, and Eclipse Help, for the Project Manager to be able to launch the viewer, you will need to add appropriate commands to `%OMSYSHOME%\common\local\config\local_omsys.ini`. See §1.3.7 [Establish system-wide configuration settings](#) on page 33.

You can specify options to display a console window while the conversion is running, and a log file when the conversion is complete. See §2.6 [Customizing the DITA2Go Project Manager](#) on page 45.

## 2.6 Customizing the DITA2Go Project Manager

You can specify several default values that will apply to all **DITA2Go** projects you subsequently create, modify, or run via the **DITA2Go** Project Manager. These default settings are on the **Preferences** tab.

*Specify a text editor*

You will need a text editor to manage project configuration files and display log files. The default editor is Windows Notepad. To replace Notepad, in the **Text editor** field enter one of the following:

- the name of your text editor, if the editor is already on your system execution path
- the full absolute path to the editor executable, if it is not on the system path.

If the path to the editor executable includes spaces, you must enclose it in double quotes.

If your desktop has a shortcut to your preferred text editor, an easy way to supply the path is as follows:

1. Right-click the desktop shortcut and choose **Properties > Shortcut**.
2. Copy the contents of the **Target** field, including any path-enclosing quotes.
3. Paste the contents into the **Text editor** field on the **Preferences** tab of the **DITA2Go** Project Manager.

*Choose what to display for a conversion*

Check whether you want either or both of the following displayed when you click **Run Project**:

- a Windows console window, while the project is running
- the resulting log file, after the run finishes.

The log file is displayed in the text editor you named; see § [Specify a text editor](#) on page 45.

The default name of every **DITA2Go** log file is `_d2g_log.txt`, and the default location is the project output directory. With a text editor you can change the name and location in the starting configuration file; see §4.2 [Logging conversion events](#) on page 74. However, you cannot use the **DITA2Go** Project Manager to change this setting. All you can do is point the **DITA2Go** Project Manager to a log file (or any other existing file). Log entries for successive runs of the same conversion are appended to the same log file.

*Choose Central  
vs. Local*

You can establish a default location for configuration files that contain settings that apply only to conversion projects for a particular DITA source document. See §39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733.

*Save and restore  
preferences*

To save the current set of options, click **Set Options**.

Click **Reset Options** to restore options to any of the following:

- The options enabled the last time you clicked **Set Options**.
- The options that were in force when you started the current **DITA2Go** Project Manager session.
- The original default options.

## 2.7 Converting documents from the command line

**DITA2Go** can be run from a command line via DCL, using the version of `dcl.exe` located in `%omsyshome%\common\bin`; see §1.3.3 [Install DITA2Go](#) on page 30.

*In this section:*

- §2.7.1 [Executing the correct version of DCL](#) on page 46
- §2.7.2 [Understanding how to run DITA2Go DCL](#) on page 47
- §2.7.3 [Creating a script to run DITA2Go DCL](#) on page 48

*See also:*

- §45 [Converting via DCL](#) on page 809

### 2.7.1 Executing the correct version of DCL

If you have **Mif2Go** version 3.3 installed on your system, you already have an older copy of DCL executable `dcl.exe` in the Windows system directory. If you need to keep that copy in place for **Mif2Go** system builds, on the command line *you must use the full path* to the newer **DITA2Go** `dcl.exe`; otherwise Windows will use the old copy instead, and you will wonder why **DITA2Go** does not work as expected.

To ensure you are executing the correct version of `dcl.exe`, create a Windows environment variable to use on the command line:

1. In *Control Panel* (on Windows XP, for example):  
**Control Panel > System > Advanced > Environment Variables**
2. If DCL is not listed in the **System variables** section, click **New** to create this environment variable. For example:

**Variable name:** DCL

**Variable value:** `%omsyshome%\common\bin\dcl.exe`

If DCL is already listed with a different value, you will have to use a different name.

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

Now you can execute DCL on the command line with the command `%DCL%` (or `%dcl%`) instead of typing the full path.

## 2.7.2 Understanding how to run DITA2Go DCL

To use `dcl.exe` to convert DITA documents:

1. Open a Windows *Command Prompt* window.
2. Navigate to the output directory where your **DITA2Go** project configuration file is located.
3. Type a command of the following form:

```
%dcl% -f output_format [-o output_file] input_file
```

where:

<i>output_format</i>	can be one of the following: HTML XML XHTML HTMLHelp JavaHelp OracleHelp EclipseHelp OmniHelp ASCII DITA DocBook Word WinHelp
<i>output_file</i>	is optional for RTF or HTML output (but required for XML output); one of: <i>name</i> - base file name (path optional), with or without extension <i>ext</i> - output file-name extension, period required
<i>input_file</i>	path (absolute or relative) to your .ditamap, .bookmap, .dita, or .xml file, with extension

4. Press **Enter**.

For example, at a command prompt:

```
D:\Tests\HTML> %dcl% -f html ..\DITATestSuite.ditamap
```

would produce HTML files in directory HTML, from map file DITATestSuite.ditamap located in directory Tests.

If your *input\_file* has an extension other than .ditamap, .bookmap, .dita, or .xml, you must include one more argument, before the `-f` switch:

```
-s xml
```

to tell `dcl.exe` what type of file you are converting.

Although `dcl.exe` provides the `-o` switch for naming output location, file, or extension, typically you do not need this switch. For topics, DITA provides a naming method for the output files: the `copy-to` attribute. If your DITA files do not use this attribute, the names of output files are based on the topic `id` attribute. If necessary, you can remap file names for HTML output; see §43.3 [Renaming output files for automated systems](#) on page 781.

## 2.7.3 Creating a script to run DITA2Go DCL

If you get tired of typing the command-line switches every time you run `dcl.exe`, you can make yourself a script that calls `dcl.exe`, using system variable `%dcl%` (see §2.7.1 [Executing the correct version of DCL](#) on page 46). You can give the script file any name (*except* `dita2go`; that name is reserved), as long as the file has extension `.bat`.

For example, to create `MyD2G.bat`:

1. Using a text editor, copy the following line and paste it into a new text file:  
`%dcl% -f%1 %2`
2. Save the new file in your **DITA2Go** output directory as "`MyD2G.bat`", and exit the text editor.

**Note:** The quote marks are not part of the file name. Their purpose is to prevent the text editor from saving the file as `MyD2G.bat.txt`, which is what many editors would do otherwise.

Now you can run `dcl.exe` from the command line as follows:

```
MyD2G output_format input_file
```

For example:

```
D:\Tests\HTML> MyD2G html ..\DITATestSuite.ditamap
```

would produce HTML files in directory `HTML`, from map file `DITATestSuite.ditamap` located in directory `Tests`.

You can customize `MyD2G.bat` for a particular type of output or for a specific project. For example, to modify `MyD2G.bat` so it always produces HTML Help:

```
%dcl% -f htmlhelp %1
```

You would run `MyD2G.bat` as follows:

```
MyD2G input_file
```

For example:

```
D:\Tests\MyHHproject> MyD2G ..\DITATestSuite.ditamap
```

Suppose you always convert a `.ditamap`; your output directory is always a subdirectory of the directory containing the `.ditamap`; and you always produce HTML 4.0 output:

```
%dcl% -f html ..\%1.ditamap
```

For a map named `ProjectA.ditamap` you would run `MyD2G.bat` as follows:

```
MyD2G input_file
```

For example:

```
D:\Maps\ToHTML> MyD2G ProjectA
```

If the only conversion you are likely to run from your output directory is from `ProjectA.ditamap`, change the command in `MyD2G.bat` to:

```
%dcl% -f html ..\ProjectA.ditamap
```

Now you can run `MyD2G.bat` as follows:

```
D:\Maps\ToHTML> MyD2G
```

See §45 [Converting via DCL](#) on page 809 for additional ways to modify and deploy DCL commands.

# 3 Editing configuration files

---

This section explains how **DITA2Go** configuration files are created, and presents the rules for adding and modifying configuration settings. Topics include:

- §3.1 [Working with DITA2Go configuration files](#) on page 49
- §3.2 [Editing files with the Configuration Manager](#) on page 49
- §3.3 [Understanding where project settings come from](#) on page 61
- §3.4 [Understanding the rules for configuration settings](#) on page 62
- §3.5 [Specifying file paths in configuration settings](#) on page 64
- §3.6 [Using wildcards in configuration settings](#) on page 65
- §3.7 [Commenting out configuration sections](#) on page 66
- §3.8 [Ending a configuration file](#) on page 66

*See also:*

- §39 [Working with templates](#) on page 727

## 3.1 Working with DITA2Go configuration files

To add or change conversion settings after you set up a **DITA2Go** 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 **DITA2Go** configuration files:

- The **DITA2Go 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 §3.2 [Editing files with the Configuration Manager](#) on page 49
- 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.

## 3.2 Editing files with the Configuration Manager

The **DITA2Go** Configuration Manager gives you access to all the configuration settings, output formats, language strings, and macros that affect a conversion project, regardless of which configuration or template files hold those values.

*In this section:*

- §3.2.1 [Understanding how to use the Configuration Manager](#) on page 50
- §3.2.2 [Starting the Configuration Manager](#) on page 52
- §3.2.3 [Setting Configuration Manager preferences](#) on page 52
- §3.2.4 [Establishing a starting point](#) on page 54
- §3.2.5 [Choosing a configuration category or file type](#) on page 54
- §3.2.6 [Understanding variable vs. fixed names and keys](#) on page 55



- §3.2.7 [Choosing the kind of change to make](#) on page 55
- §3.2.8 [Selecting a configuration section](#) on page 59
- §3.2.9 [Selecting a configuration setting](#) on page 60
- §3.2.10 [Selecting a configuration file](#) on page 60
- §3.2.11 [Specifying a final value](#) on page 61

## 3.2.1 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:*

- §3.2.1.1 [Drilling down to find a section or setting](#) on page 50
- §3.2.1.2 [Heeding status and result messages](#) on page 51
- §3.2.1.3 [Getting help with controls and configuration data](#) on page 51
- §3.2.1.4 [Providing help for your own format and macro definitions](#) on page 51
- §3.2.1.5 [Correcting configuration errors](#) on page 51

### 3.2.1.1 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, formats, language 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.

For example, suppose you wish to change the bullet style for unordered-list elements that are mapped to the *Bulleted2* output format, for the document involved in your current project, and for any related conversion projects that work on this document. You would follow these steps:

1. Start the Configuration Manager and select your project.
2. Because *Bulleted2* is an paragraph format for an unordered-list item, choose configuration category **Unordered** (under **List Formats**, under **Paragraph Formats**, under **Text Formats**).
3. Choose the action to perform: **Edit one setting in section**.
4. Choose the section where the setting occurs: **Bulleted2**.
5. Choose **list style** from the list of settings that are present in some or all definitions of *Bulleted2* that can affect your project.
6. Choose, from a list of the format configuration files accessed by your project, the file that is specific to the document for this project.



7. Choose a different value for **list style**.
8. Click **Finish action**, and you are done.

### 3.2.1.2 Heading 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.

### 3.2.1.3 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 **DITA2Go User's Guide** that explains that selection or control.

**Note:** Make sure that `ugdita2go.chm`, the HTML Help version of the **DITA2Go User's Guide**, is available in `%OMSYSHOME%\d2g\usersguide`; see §1.3.3.2 [Finish installing DITA2Go](#) on page 31.

### 3.2.1.4 Providing help for your own format and macro definitions

The Configuration Manager displays one-line descriptions of all fixed-name configuration sections, including all format and format component definitions and all macro definitions supplied in the **DITA2Go** distribution.

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

No Help for this section

*Help for format definitions*

To provide descriptive help for a format, include a setting in the definition of that format for keyword `help`, and supply descriptive text as the value; for example:

```
[OverTitle]
based = PrefaceTitle
help = Format for title of Overview topic
```

*Help for macro definitions*

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
```

### 3.2.1.5 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.

**Note:** Format names (other than paragraph format names) that do not end with the required suffix always show red in the Configuration Manager. You cannot use the Configuration Manager to rename formats, because renaming a format can affect many other types of settings.

*Format names  
without a valid  
suffix*

If a format name is displayed in red on the **Section** page, usually this is because either the name lacks an appropriate suffix, or the name ends in a suffix that is not valid in the file where the format is defined. [Table 7-1](#) on page 114 shows valid suffixes for format names.

The Configuration Manager assumes any format in a `Formats` configuration file is a paragraph format, unless its name ends in `Char`. Any format in a `Tables` configuration file is assumed to be a table format, unless its name ends in `Row` or `Cell`.

To correct an improperly named format that is already in use, besides renaming the format, also assign the new name to the old in `[FormatAliases]`; see §7.3.2 [Mapping legacy names to defined formats](#) on page 113. In the Configuration Manager, select the top-level category for the type of format: **Text Formats**, **Table Formats**, **Page Formats**, or **Format Components**.

For example, if you are changing the name of a special table-row format so it ends in `Row`, select category **Table Formats**, action **Add new setting to section**, and section `FormatAliases`; add a setting such as the following:

```
[FormatAliases]
; Old format name = new name with proper suffix
MyTableRow1 = MyTableFirstRow
```

This will ensure **DITA2Go** substitutes the new name for the old, in case the old name persists in some settings.

### 3.2.2 Starting the Configuration Manager

To access configuration values for your project, start the **DITA2Go** Configuration Manager, either of the following ways:

- **On the Windows desktop:** double-click the shortcut to `%OMSYSHOME%\common\bin\d2gcm.exe`. (If you do not already have a desktop shortcut to this program, see §1.3.3.2 [Finish installing DITA2Go](#) on page 31.) The Configuration Manager opens to the **Start** page; see §3.2.4 [Establishing a starting point](#) on page 54.
- **From the DITA2Go Project Manager:** with your project selected, click **Configure** (see §2.4 [Inspecting and editing configuration files](#) on page 44). The Configuration Manager opens to the **Category** page; see §3.2.5 [Choosing a configuration category or file type](#) on page 54.

Before you use the Configuration Manager to change the way your projects work, consider visiting the **Preferences** page; see §3.2.3 [Setting Configuration Manager preferences](#) on page 52.

### 3.2.3 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:*

- §3.2.3.1 [Specifying colors for different types of settings](#) on page 53
- §3.2.3.2 [Annotating changes made to configuration files](#) on page 53

### 3.2.3.1 Specifying colors for different types of settings

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

<b>Available fixed-name sections</b>	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.
<b>Available fixed-key settings</b>	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.
<b>Variable names and keys</b>	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.
<b>Internal defaults</b>	Values that <b>DITA2Go</b> uses for settings that do not appear in any of the files for the current session.
<b>System configuration files</b>	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 §39.4.1 <a href="#">Understanding what configuration files are available</a> on page 735.

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 §3.2.1.5 [Correcting configuration errors](#) on page 51.

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

### 3.2.3.2 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=DITA2Go User's Guide
;2012-11-30 15:16:08: CS deleted duplicate Title
!=Title=DITA2Go User's Guide
```

Annotation takes effect immediately.

### 3.2.4 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.

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 §39.2 [Referencing configuration files and templates](#) on page 731) 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 §3.2.5 [Choosing a configuration category or file type](#) on page 54.

### 3.2.5 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 §3.2.4 [Establishing a starting point](#) on page 54.

*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>
<b>General Configuration</b>	Project configuration options and settings	<a href="#">4</a>
<b>Text Formats</b>	Definitions of paragraph and character output formats	<a href="#">7.6</a>
<b>Table Formats</b>	Definitions of table output formats	<a href="#">7.7</a>
<b>Page Formats</b>	Page layout definitions for RTF output	<a href="#">7.8</a>
<b>Format Components</b>	Building blocks for link and other format definitions	<a href="#">8</a>
<b>Languages</b>	Localization strings for output formats	<a href="#">8.9</a>
<b>Macros</b>	Definitions of <b>DITA2Go</b> macros	<a href="#">37</a>
<b>Content Models</b>	Configuration-style representation of a DTD	<a href="#">41</a>

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

- + Text Formats
  - + Paragraph Formats
    - + Title/Heading Formats
  - + List Formats
    - + Ordered
    - + Unordered
  - ...

*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. For example, if you want to edit one or more of the many *Bulleted* formats, and you want to see them all, you could select category **Unordered List Formats** and specify *Bulleted\** for **Section matches**. 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 §3.2.7 [Choosing the kind of change to make](#) on page 55.

### 3.2.6 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 format definitions and macro definitions, where the section name is the name of the format or macro.
- A **fixed-name configuration section** has a name defined by **DITA2Go**; examples include Setup and HTMLOptions.

*Variable-key vs.  
fixed-key settings*

- **Variable-key settings** are characterized by keys that usually consist of a format name (such as the settings in section HTMLParaStyles) or an object identifier (such as the settings in section GraphGroup).
- **Fixed-key settings** must use a key from a set of **DITA2Go**-specified names for keys that are valid in their section (such as the settings in section Setup). See §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769.

### 3.2.7 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 §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769.

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 §3.2.7.8 [Merging duplicate sections or settings](#) on page 59. You can change it back again, but the Configuration Manager will continue to nag you about duplicates until you resolve them.

*In this section:*

- §3.2.7.1 [Adding a new section or setting](#) on page 56
- §3.2.7.2 [Editing a section or setting](#) on page 56
- §3.2.7.3 [Deleting a section or setting](#) on page 57
- §3.2.7.4 [Restoring a deleted section or setting](#) on page 57
- §3.2.7.5 [Renaming a section or setting](#) on page 58
- §3.2.7.6 [Moving a section or setting](#) on page 58
- §3.2.7.7 [Copying a section or setting to another configuration file](#) on page 59
- §3.2.7.8 [Merging duplicate sections or settings](#) on page 59

### 3.2.7.1 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:

<b>Add new fixed-name section to file</b>	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 <b>Section</b> page.
<b>Add new variable-name section to file</b>	To add a format definition or a macro definition, select this option. You get to specify the name on the <b>Section</b> page.
<b>Add new setting to section</b>	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>
<b>Add new variable-key setting to one section</b>	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 §3.2.7.2 [Editing a section or setting](#) on page 56.

**Note:** You cannot add a setting to a named macro. Macro sections do not contain settings. Instead, choose **Edit full section content**; see §3.2.7.2 [Editing a section or setting](#) on page 56.

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.

### 3.2.7.2 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:

<b>Edit full section content</b>	Edit the content of a section, and also the section heading and any comments; you can even change the name of the section.
<b>Edit one setting in section</b>	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.



### 3.2.7.3 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:

<b>Delete section from one file</b>	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.
<b>Delete setting from one section</b>	First you choose a section, then you pick the setting, then you choose the file from which you want to delete the setting. <i>Do not use for named macros</i> ; 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.

### 3.2.7.4 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:

<b>Restore deleted section in one file</b>	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 (§3.2.7.1 <a href="#">Adding a new section or setting</a> on page 56) or copy it from another file (§3.2.7.7 <a href="#">Copying a section or setting to another configuration file</a> on page 59).
<b>Restore deleted setting in one section</b>	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.

### 3.2.7.5 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:

<b>Rename variable-name section in one file</b>	Only variable-name sections, such as format and macro definitions, can be renamed.
<b>Rename variable-key setting in one section</b>	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.

### 3.2.7.6 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:

<b>Move section within one file</b>	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.
<b>Move section between files</b>	You might want to move a section “upstream” or “downstream”; that is, make it apply more widely or less widely.
<b>Move setting within one section</b>	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.
<b>Move setting between files</b>	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.



### 3.2.7.7 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.

### 3.2.7.8 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.

## 3.2.8 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 §3.2.7 [Choosing the kind of change to make](#) on page 55.

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

For example, if you had selected category **Para Char Formats** and specified **Bulleted\*** for **Section matches**, you might see this list of sections:

```
Bulleted1
Bulleted1L2
Bulleted1L3
Bulleted2
Bulleted2L3
BulletedL
```

Click a section name to select it. Immediately below the selection box you see a statement that characterizes the section. For example, if you had selected `Bulleted2` (or any other name) from the section list above, you would see:

Section is in current inis and is variable-name with fixed keys

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 §3.2.10 [Selecting a configuration file](#) on page 60

---

**Setting** The Configuration Manager switches to the **Setting** page; see §3.2.9 [Selecting a configuration setting](#) on page 60.

---

### 3.2.9 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 §3.2.7 [Choosing the kind of change to make](#) on page 55. 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. For example, if you had selected `Bulleted2` from the list on the **Section** page, you would see a list of settings that are valid in this paragraph-format-definition section.

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 §3.2.3 [Setting Configuration Manager preferences](#) on page 52) 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 §3.2.10 [Selecting a configuration file](#) on page 60.

#### 3.2.10 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 §3.2.7 [Choosing the kind of change to make](#) on page 55. 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 §3.2.3 [Setting Configuration Manager preferences](#) on page 52), 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 §3.2.11 [Specifying a final value](#) on page 61.

### 3.2.11 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 §3.2.7 [Choosing the kind of change to make](#) on page 55. 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 §3.2.7 [Choosing the kind of change to make](#) on page 55. 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.

## 3.3 Understanding where project settings come from

When you set up a new conversion project, the **DITA2Go** Project Manager copies a new output-type-specific starting configuration file into your project directory. This file is populated with the settings you specify at set-up time. **DITA2Go** gets this file from a repository of configuration templates located in your **DITA2Go** distribution; see §39.1.1 [Understanding how templates are organized](#) on page 727. Each configuration template already contains values for basic settings specific to the output type for your project.

*Default  
configuration  
values*

Configuration values present in **DITA2Go** configuration files at set-up time 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 **DITA2Go** 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 **DITA2Go**; 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
- a required chain of configuration templates located in your **DITA2Go** 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 §39.1 [Working with configuration templates](#) on page 727.

## 3.4 Understanding the rules for configuration settings

Every **DITA2Go** 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 **DITA2Go**, 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 **DITA2Go** rules:

- **The first line in the file must be a comment**; **DITA2Go** 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), **DITA2Go** 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](#)  
[Multiple values are separated by spaces](#)

	<p>Spaces and tabs: some retained, some removed</p> <p>Comments start with a semicolon</p> <p>Boolean values can be expressed various ways</p>
Section names must be unique	Section names must be unique. If you use the same section name twice in your configuration file, <i>only the first section is processed</i> . Otherwise, order of sections does not matter, except for macro sections (see §37.1.1.2 <a href="#">Understanding where you can define named macros</a> on page 680).
Key names must be unique in a section	Each <code>key=</code> 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	<p>All ASCII characters are valid in key names, with the following exceptions:</p> <ul style="list-style-type: none"> <li>“?” and “*” are treated as wildcards, unless you turn off wildcard usage; see §4.1.10 <a href="#">Specifying how to treat cases, spaces, and wildcards</a> on page 73. (However, when you override a configuration setting with a configuration variable, <b>DITA2Go</b> does not recognize wildcards in the key name; see §42.2.4 <a href="#">Assigning values to configuration variables</a> on page 768.)</li> <li>“;” or “[” must be prefixed with escape character “\” if you want to <i>start</i> a key name with either of these characters.</li> </ul> <p>Spaces are nominally allowed in key names, but the spaces are ignored unless you turn off <code>[Options]SpacelessMatch</code>; see §4.1.10 <a href="#">Specifying how to treat cases, spaces, and wildcards</a> on page 73. Do not use spaces if you can possibly avoid them.</p>
Key names are not case sensitive, by default	Comparisons of key names are caseless, unless you turn on case sensitivity; see §4.1.10 <a href="#">Specifying how to treat cases, spaces, and wildcards</a> on page 73. (However, when you override a configuration setting with a configuration variable, the key name <i>is</i> case sensitive; see §42.2.4 <a href="#">Assigning values to configuration variables</a> on page 768.)
Fixed-key sections differ from variable-key sections	Configuration files contain two kinds of sections: those with <i>fixed keys</i> (key names predefined by <b>DITA2Go</b> ) and those with <i>variable keys</i> . For example, sections such as <code>[HTMLOptions]</code> and <code>[WordOptions]</code> are for settings with fixed key names; sections such as <code>[HTMLParaStyles]</code> and <code>[HelpStyles]</code> are for settings with key names you specify, typically names of formats assigned to DITA elements.
Order of settings can be important for variable keys	<p>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 §3.6 <a href="#">Using wildcards in configuration settings</a> on page 65).</p> <p>Often the variable-key names you specify are names of formats assigned to DITA elements, such as paragraph, character, or table formats.</p>
Multiple values are separated by spaces	Some variable-key sections allow multiple values for each key: sections such as <code>[HTMLParaStyles]</code> , <code>[WordStyles]</code> , and <code>[HelpStyles]</code> , where you can assign multiple properties to each format. Use spaces between values.
Spaces and tabs: some retained, some removed	<p><b>DITA2Go</b> treats spaces and tabs in configuration settings as follows:</p> <ul style="list-style-type: none"> <li>Spaces and tabs before the <code>key</code> and before the equals sign are ignored, unless <code>[Options]SpacelessMatch=No</code>, in which case those before the <code>key</code> are <i>not</i> ignored (see §4.1.10 <a href="#">Specifying how to treat cases, spaces, and wildcards</a> on page 73).</li> <li>If the equals sign is followed by one or more spaces or tabs, the first such space or tab is removed, and <i>the rest are treated as part of the value</i>. Put no more than one space after the equals sign. If you want to align settings vertically for readability, put extra spaces <i>before</i> the equals sign, not after.</li> </ul>



- 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 **DITA2Go** 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). **DITA2Go** 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 §3.7 [Commenting out configuration sections](#) on page 66.

*Boolean values can be expressed various ways*

For an On/Off value, **DITA2Go** recognizes “1” (numeral one), “Yes”, and “True” as On, and “0” (zero), “No”, and “False” as Off.

## 3.5 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 26, 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 §43.1 [Executing operating-system commands](#) on page 777
- Windows command parameters; for example, see §44.11 [Archiving deliverables](#) on page 803.

In Windows API calls, forward slashes work fine, because the original Windows programmers compiled Windows on VAX/VMS machines.

**Note:** XML catalog-key path settings *must* use forward slashes. See §4.1.1 [Connecting to XML catalogs](#) on page 67

*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 26.

*Most relative paths relate to the project directory*

Most path settings (other than those listed in [Table 3-1](#) on page 65) can be either relative or 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 3-1](#) on page 65
- [Automation]ShipPath, which is relative to the wrap directory; see §44.3 [Understanding path values for deliverables](#) on page 788.

*Paths to configuration templates should be absolute*

Settings that reference configuration templates, or other files in the **DITA2Go** distribution directory structure, should use absolute paths that begin with environment variable %OMSYSHOME%; for example:

```
Configs = %omsysHOME%\d2g\local\config\local_d2htm_config.ini
```

See §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29.

*Some relative paths relate to the configuration file location*

If you specify a relative path in any of the settings listed in [Table 3-1](#) on page 65, 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 §39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733) and the project directory are immediately below the source directory. If you have files all over the place, portability becomes impossible.

**Table 3-1: Absolute vs. relative file-path settings**

Section	Setting	Absolute or relative file path?	Ref.
[Templates]	<i>All settings</i>	Paths to templates in the <b>DITA2Go</b> distribution should be absolute and start with %OMSYSHOME%	<a href="#">39.1</a>
		Paths to your own template files should be relative to the location of the configuration file in which the setting occurs	<a href="#">39.5</a>
[Setup]	DTDPath	Absolute path recommended	<a href="#">4.1.3</a>
[Logging]	LogFileName	Keep setting in project configuration file	<a href="#">4.2</a>
	HistoryFileName	Keep setting in project configuration file	<a href="#">4.2</a>
	EditorFileName	Absolute path required	<a href="#">4.2</a>
[ConditionOptions]	DitavalFile	Absolute path recommended	<a href="#">9.1.1</a>
[OmniHelpOptions]	ProjectTemplate	Keep setting in project configuration file	<a href="#">19.5.7</a>

*Paths to other applications must be absolute*

File paths to non-**DITA2Go** 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.

## 3.6 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, except in element paths, where it can appear anywhere in the path.

You can use wildcards whenever the key is a format name or an identifier, provided you have not turned off wildcard usage (see §4.1.10 [Specifying how to treat cases, spaces, and wildcards](#) on page 73). For example, to make all 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.

**DITA2Go** applies the first entry in a section that matches for each key name, so always put the exceptions before the general case.

## 3.7 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 **DITA2Go**, 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.

## 3.8 Ending a configuration file

All configuration files and templates in your **DITA2Go** 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.



# 4 Setting basic conversion options

---

This section explains how to use basic **DITA2Go** configuration settings to convert documents from DITA XML to other representations. Topics include:

§4.1 [Specifying operating settings](#) on page 67

§4.2 [Logging conversion events](#) on page 74

§4.3 [Identifying files and elements](#) on page 76

§4.4 [Processing graphics](#) on page 77

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

§15 [Converting to print RTF](#) on page 219

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

§16 [Producing on-line Help](#) on page 243 *through*

§21 [Generating Eclipse Help](#) on page 413

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

§22 [Converting to HTML/XHTML](#) on page 429 *through*

§36 [Marking HTML table cells for WAI](#) on page 667

## 4.1 Specifying operating settings

*In this section:*

§4.1.1 [Connecting to XML catalogs](#) on page 67

§4.1.2 [Accommodating specializations](#) on page 68

§4.1.3 [Specifying a DITA XML DTD](#) on page 68

§4.1.4 [Generating a map from a DITA topic file](#) on page 69

§4.1.5 [Accommodating paths to network drives](#) on page 70

§4.1.6 [Checking output type and file extension](#) on page 70

§4.1.7 [Producing print output selectively](#) on page 71

§4.1.8 [Including element tags and paths in output](#) on page 72

§4.1.9 [Reusing or discarding ASCII DCL files](#) on page 73

§4.1.10 [Specifying how to treat cases, spaces, and wildcards](#) on page 73

*See also:*

§3.4 [Understanding the rules for configuration settings](#) on page 62

### 4.1.1 Connecting to XML catalogs

By default, **DITA2Go** uses XML catalogs to resolve paths to resources. Because DITA 1.2 depends on catalogs so heavily, your DITA document should use XML catalogs to locate DTD, system entity, or stylesheet files.

To specify keys to each XML catalog:

```
[Catalogs]
; UseCatalogs = Yes (default, resolve paths to resources
; via specified XML catalogs) or No
UseCatalogs = Yes
; CatalogKeys = list of keys to local XML catalogs
CatalogKeys = dital.2 dital.1
```

Catalog key names are arbitrary. Several catalog keys are already defined in site-wide system file %OMSYSHOME%\common\system\config\omsys.ini. Put your own catalog key names in %OMSYSHOME%\common\local\config\local\_omsys.ini, because the corresponding system file is replaced during updates.

To define each catalog key:

```
[Catalogs]
catalogkey = path/to/local/catalog.xml
; Use forward slashes, not backslashes in all these paths:
dita1.1 = %omsyshome%/d2g/dtds/dita1.1/catalog-dita.xml
; Use this key instead of dita1.1 if you want 1.2 features:
dita1.2 = %omsyshome%/d2g/dtds/dita1.2/catalog-dita.xml
; This key supports the DITA4Publishers specialization:
d4pubs = %omsyshome%/d2g/specializations/d4pubs/dtds/d2g_catalog.xml
```

**Note:** Use only forward slashes in catalog paths, not backslashes.

For example, in local\_omsys.ini:

```
[Catalogs]
mydoc = D:/mydtds/mydoctypes/catalog.xml
myxml = D:/mydtds/myxmldomain.doctypes/dtd/catalog.xml
```

And in your project configuration file:

```
[Catalogs]
CatalogKeys = mydoc myxml dita1.1
```

You can define all the catalog keys for any resource you might ever want in top-level configuration file local\_omsys.ini (see §1.3.7 [Establish system-wide configuration settings](#) on page 33), then use whichever keys are appropriate on a per-project basis, in whatever order you require for each project.

## 4.1.2 Accommodating specializations

Although you can specify paths to catalogs located anywhere on your local system, if you are using specializations, you need to create your own XML catalog for each, and put that catalog where your catalog key says it is; that can be a subdirectory of its own under the DTDs. In the **DITA2Go** distribution, the catalogs are located with other DTDs.

**DITA2Go** does not have any requirements related to specialization. If you specialize the DTDs, **DITA2Go** reads your new DTDs and just works. For a tutorial on specialization, see:

<http://xiruss.org/tutorials/dita-specialization/>

*DITA For  
Publishers*

If you have already installed the DITA For Publishers plug-in for the DITA Open Toolkit, as an alternative to the d4pubs key defined in \common\system\config\omsys.ini, you can do the following:

- Copy d2g\_catalog.xml into the plugins directory where you unzipped dita4publishers
- Define environment variable DITA4PUBS
- In [Catalogs] specify the following path for the d4pubs key:  
d4pubs = %dita4pubs%/d2g\_catalog.xml

## 4.1.3 Specifying a DITA XML DTD

You need to tell **DITA2Go** where to find the DTD for your document only if you are not using catalogs, and your document includes specializations. If the SystemID in the

doctype declaration for your DITA XML document does not designate an accessible local “file:” URL, you must specify a local path to the DTD.

To tell **DITA2Go** where to find the DTD for your project:

```
[Setup]
; DTDPath = local path to use for DITA DTDs if the SystemID in the
; doctype declaration is not an accessible local "file:" URL
DTDPath=%OMSYSHOME%\d2g\dtlds\dita1.1
```

A copy of the Oasis DITA 1.1 DTD is included in your **DITA2Go** distribution, in directory %omsysHOME%\common\dtlds\dita1.1. If you have specializations, you must use your own DTDs instead. Or if you already have your own copies of the Oasis DTDs elsewhere for use with other applications, you can use those instead.

*See also:*

§4.1.1 [Connecting to XML catalogs](#) on page 67

#### 4.1.4 Generating a map from a DITA topic file

**DITA2Go** can work directly on a DITA topic file without a map that references the topic file. However, output would not include a TOC, and related-links would not be resolved. Therefore, if a map is not already present, **DITA2Go** generates a map for your project, unless you do not want a map.

*In this section:*

§4.1.4.1 [Choosing whether to generate a map](#) on page 69

§4.1.4.2 [Specifying options for a generated map](#) on page 69

##### 4.1.4.1 Choosing whether to generate a map

By default, when you specify a DITA topic file as input, if no map that references that topic file is present, **DITA2Go** generates a map for your project.

To prevent **DITA2Go** from generating a map:

```
[MapGeneration]
; GenerateMapIfMissing = Yes (default, if no map found, generate a
; ditamap with the same name and run that instead), or No
GenerateMapIfMissing = No
```

When GenerateMapIfMissing=Yes, **DITA2Go** first looks for a map with the same name as the topic file, in the same directory; if found, **DITA2Go** processes that map. If the directory contains both a .ditamap and a .bookmap, **DITA2Go** uses the .bookmap. **DITA2Go** never overwrites an existing map file.

When GenerateMapIfMissing=No, **DITA2Go** processes the topic file by itself. The output will not include a TOC, and related-links will not be resolved.

##### 4.1.4.2 Specifying options for a generated map

To specify options for a **DITA2Go**-generated map:

```
[MapGeneration]
; MapRootElem = root element to use for map, default "map"
MapRootElem = map
; MapTitleElem = title element to use for map, default "title"
MapTitleElem = title
; MapTitle = title element content for map, default is no title
MapTitle = Title content for map
; UseTopicShortdesc = Yes (default, include shortdescs from topic)
; or No (exclude text in the topics from the map).
```

```

UseTopicShortdesc = Yes
; MapLanguage = value of xml:lang on map root element, default en-us
MapLanguage = en-us
; PublicID = PUBLIC ID to use for map DOCTYPE, default as below
PublicID= "-//OASIS//DTD DITA 1.1 Map//EN"
; SystemID = SYSTEM ID to use for map DOCTYPE, default as below
SystemID= "http://docs.oasis-open.org/dita/v1.1/OS/dtd/map.dtd"

```

You also use these options when you run command-line utility **DITA2Map** as a stand-alone program. **DITA2Map** simply generates a map and stops there, without producing any further output. See §46 [Creating a map with DITA2Map](#) on page 815.

### 4.1.5 Accommodating paths to network drives

If parts of your DITA source reside on different drives, **DITA2Go** cannot determine relative paths from the project directory to those parts, and must use absolute paths instead.

To direct **DITA2Go** to use absolute rather than relative paths:

```

[Options]
; UseFullPath = No (default, use relative paths to project components)
; or Yes (always use absolute paths)
UseFullPath = Yes

```

When `UseFullPath=No`, **DITA2Go** determines the relative path from the project directory to the top-level map. **DITA2Go** continues this process, determining the relative path from the project directory to each component, for essentially every path in the maps and topics. If the initial map is on a different drive from the drive where the project directory is located, **DITA2Go** cannot determine a relative path, and uses no path at all. This can cause your conversion project to fail.

When `UseFullPath=Yes`, you get absolute paths to images as well as to map and topic files; this means that for HTML output, references to graphics might resolve only on the original system. To get around this problem, you must either remove paths from graphics references or specify a path explicitly. (This is not an issue for RTF output, where **DITA2Go** embeds images in the RTF code itself when possible, or places graphics files in the same directory as the RTF files.)

To remove path information from references to images for HTML output:

```

[Graphics]
; StripGraphPath = No (default)
; or Yes (remove path from referenced graphics)
StripGraphPath = Yes

```

To specify a path to graphics files for HTML output:

```

[Graphics]
; GraphPath = path to use (replacing any previous) for all graphics
GraphPath = path/to/graphics/files

```

See §32.1 [Locating graphics files for HTML](#) on page 611.

### 4.1.6 Checking output type and file extension

The project configuration file that **DITA2Go** Project Manager copied to your output directory when you set up your conversion project already contains the correct setting for the output type (which is indicated by the configuration file name). *Do not change this setting.*

Table 4-1 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 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 4-1 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	_d2eclipse.ini	<a href="#">21</a>
	Microsoft HTML Help	.htm	_d2htmlhelp.ini	<a href="#">18</a>
	JavaHelp	.htm	_d2javahelp.ini	<a href="#">20</a>
	OmniHelp	.htm	_d2omnihelp.ini	<a href="#">19</a>
	Oracle Help for Java	.htm	_d2oraclehelp.ini	<a href="#">20</a>
HTML	Standard HTML	.htm	_d2html.ini	<a href="#">22</a>
	XHTML	.xhtml	_d2xhtml.ini	<a href="#">22</a>
XML	DITA XML	.dita ( <i>not settable</i> )	_d2dita.ini	<a href="#">24</a>
	Docbook XML	.ent	_d2docbook.ini	<a href="#">26</a>
	Generic XML	.xml	_d2xml.ini	<a href="#">23</a>
RTF	WinHelp	.rtf ( <i>not settable</i> )	_d2winhelp.ini	<a href="#">17</a>
	Print RTF	.rtf	_d2rtf.ini	<a href="#">15</a>

### 4.1.7 Producing print output selectively

DITA topicref attribute @print (with value yes, no, or printonly) allows you to specify whether the referenced topic is to be included in print output, excluded from print output, or included only in print output.

By default, **DITA2Go** includes topics referenced with @print="printonly" in RTF output for Word or WordPerfect, but in no other output type. To include topics for which @print="printonly" in other output types:

```
[Setup]
; PrintProject = No (default) or Yes (override topic inclusion
; that is based on value of @print attribute)
PrintProject = Yes
```

When PrintProject=Yes, topics for which @print=printonly are included in output.

When PrintProject=No, topics for which @print="printonly" are included only in Word or WordPerfect output; and topics for which @print="no" are included in all outputs, except for Word and WordPerfect.

Table 4-2 shows the effect of the PrintProject setting for different values of the @print attribute.

**Table 4-2: Topics included in output based on PrintProject setting**

PrintProject	@print attribute	Include topic in output?
No	<i>not present</i>	Yes
	yes	Yes
	no	No for print RTF, Yes for all others
	printonly	Yes for print RTF, No for all others
Yes	<i>not present</i>	Yes
	yes	Yes
	no	No
	printonly	Yes

**Note:** You do not need to specify a PrintProject setting at all unless you are trying to override normal expectations for type of output.

*Do it with PI markers*

If your DITA maps do not include values for the topicref @print attribute, you can achieve the same effect with PI markers inserted in the topicrefs, and a ditaval file. For example:

```
<?dtall Print="printonly" ?>
```

In this example the effect on **DITA2Go** output is the same as if you had included @print="printonly" in the topicref for each such topic.

### 4.1.8 Including element tags and paths in output

By default, **DITA2Go** includes several items from your DITA XML files in ASCII DCL, the intermediate representation of your document from which the final output is generated. These items include the original element tags and attributes, and the path used to select an output type for each element. These settings can be useful for project debugging.

To omit element tags and attributes from DCL output:

```
[ElementOptions]
; IncludeElementTags = Yes (default, include original tags and
; attributes in the DCL, for possible downstream use) or No
IncludeElementTags = No
```

To omit format selection paths from DCL output:

```
[ElementOptions]
; ShowElementPath = Yes (default, include report of element path
; used to select each format in DCL after the format call) or No
ShowElementPath = No
```

When ShowElementPath=Yes, the position of the element from which **DITA2Go** derived each paragraph or character span is written to DCL just ahead of the paragraph or span, enclosed in square brackets. This can be helpful when you specify an element path in [BlockFormatMaps] or [InlineFormatMaps], and the setting does not work as intended. You can see what **DITA2Go** considered the actual element path to be.

If you plan to keep the intermediate ASCII DCL file for further inspection or processing you must run dcl.exe twice: first to generate the intermediate ASCII DCL file, and second to generate the final output; see §2.7 [Converting documents from the command line](#) on page 46. However, you can direct **DITA2Go** to include the format selection paths in final output, also.

To include format selection paths in final output:

```
[ElementOptions]
; DisplayElementPath = No (default) or Yes(display the element path
; used to select each format at the start of the text in that format)
DisplayElementPath = Yes
```

When `DisplayElementPath=Yes`, any element-path information collected in DCL is included in final output, just ahead of the paragraph or span. `DisplayElementPath` is effective only when `ShowElementPath=Yes`.

To specify a character format for `DisplayElementPath`:

```
[ElementOptions]
; ElementPathFormat = inline format used for element path display
ElementPathFormat = CharFmt
```

### 4.1.9 Reusing or discarding ASCII DCL files

You can use **DITA2Go** to convert the files in your document to ASCII DCL, specifying `-t dcl` on the command line (see §45.3 [DCL command-line syntax](#) on page 810). Next time you run the same conversion, you can direct **DITA2Go** 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, **DITA2Go** 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`, **DITA2Go** deletes both `.dcl` and `.dcb` files from your project directory before conversion.

### 4.1.10 Specifying how to treat cases, spaces, and wildcards

You can choose how **DITA2Go** interprets 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 §3.6 [Using wildcards in configuration settings](#) on page 65.

## 4.2 Logging conversion events

Whenever you convert a document, by default **DITA2Go** 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, **DITA2Go** 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, history file, or log text editor:

```
[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 = d2g_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 = _d2g_history.txt
```

The default name of the log file is `_d2g_log.txt`, and the default name of the history file is `_d2g_history.txt`. Unless you specify a different path for `LogFileName` or for `HistoryFileName`, **DITA2Go** 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 **DITA2Go** 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, the **DITA2Go** Project Manager 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 **DITA2Go** Project Manager, if any warnings or errors occur, the Project Manager 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 **DITA2Go** from the command line, each **DITA2Go** 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 **DITA2Go** 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, **DITA2Go** 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 DITA file and the identity of the software module doing the processing. Unless you specify otherwise **DITA2Go** 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 undefined formats*

By default, **DITA2Go** logs a warning about any format that is referenced but not defined. If you get tired of seeing warnings about undefined formats:

```
[Logging]
; ShowUndefinedFormats = Yes (default) or No
ShowUndefinedFormats = No
```

#### *Recording configuration chains*

In addition to logging conversion events, you can have **DITA2Go** 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, **DITA2Go** 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:

```
Il: Ini chain for Configs:
Il:   _d2omnihelp.ini
Il:   ..\_config\d2gug_htm_document.ini
Il:   ..\_config\d2gug_document.ini
Il:   g:\omnisys\d2g\local\config\local_d2omnihelp_config.ini
Il:   g:\omnisys\d2g\system\config\d2omnihelp_config.ini
Il:   g:\omnisys\d2g\local\config\local_d2help_config.ini
Il:   g:\omnisys\d2g\system\config\d2help_config.ini
Il:   g:\omnisys\d2g\local\config\local_d2htm_config.ini
Il:   g:\omnisys\d2g\system\config\d2htm_config.ini
Il:   g:\omnisys\d2g\local\config\local_d2g_config.ini
Il:   g:\omnisys\d2g\system\config\d2g_config.ini
Il:   g:\omnisys\common\local\config\local_omsys.ini
Il:   g:\omnisys\common\system\config\omsys.ini
Il: Ini chain for Languages:
Il:   g:\omnisys\d2g\local\lang\local_d2g_lang_en.ini
Il:   g:\omnisys\d2g\system\lang\d2g_lang_en.ini
Il: Ini chain for Formats:
```

```

I1:    ..\_config\d2gug_htm_formats.ini
I1:    g:\omnisys\d2g\local\formats\local_d2htm_formats.ini
I1:    g:\omnisys\d2g\system\formats\d2htm_formats.ini
I1: Ini chain for SubFormats:
I1:    ..\_config\d2gug_htm_subformats.ini
I1:    g:\omnisys\d2g\local\formats\local_d2htm_subformats.ini
I1:    g:\omnisys\d2g\system\formats\d2htm_subformats.ini

```

This output shows the chain of general configuration files and templates referenced from starting configuration file `_d2omnihelp.ini`, for a project to generate OmniHelp from DITA.

## 4.3 Identifying files and elements

To maintain interfile and intrafile references, **DITA2Go** automatically generates unique IDs for every element and output file. These generated IDs wind up in the output as the link targets. You can specify how these IDs are created.

*In this section:*

- §4.3.1 [Generating document-wide unique IDs](#) on page 76
- §4.3.2 [Specifying building blocks for unique IDs](#) on page 76
- §4.3.3 [Specifying a prefix for bookmarks in RTF output](#) on page 77
- §4.3.4 [Checking for duplicate IDs](#) on page 77

### 4.3.1 Generating document-wide unique IDs

By default, **DITA2Go** augments element IDs with topic, file, and path information. To omit this optional information and include just the element IDs themselves:

```

[IDOptions]
; GenerateUIDs = Yes (default, augment element IDs with optional
; topic, file, and path information), or No (use element ID as is)
GenerateUIDs=No

```

When `GenerateUIDs=Yes`, **DITA2Go** combines several building blocks to generate a unique document-wide ID for every element. You can specify the building blocks to use; see §4.3.2 [Specifying building blocks for unique IDs](#) on page 76.

### 4.3.2 Specifying building blocks for unique IDs

The following three options determine what building blocks are added to an element ID to create a document-wide unique ID. To omit a building block, set its option to `No`:

```

[IDOptions]
; IDTopic = Yes (default, include topic ID in UID) or No
; IDFile = Yes (default, include file name in UID) or No
; IDPath = Yes (default, include path to file in UID) or No

```

The default is to include all three in the element ID: path, file name, and topic ID. Depending on your situation, you may be able to eliminate some building blocks. For example, if you know your DITA file names are always unique (as when using a CMS), that alone may be sufficient without paths and IDs.

The following settings determine the separators used between building blocks. To eliminate use of a separator, set its option to blank.

```

[IDOptions]
; IDPathSep = string used to replace the "/"s in the doc path
IDPathSep=P

```

```

; IDUpDir = string used to replace any "../"s at start of path
IDUpDir=U
; IDTopSep = string used to replace the "#" before the topic ID
IDTopSep=T
; IDElemSep = string used to replace the "/" between topic ID and
; element ID
IDElemSep=E

```

For example, using the default separator characters, this string:

```
UconceptsPc_methodTmainmethodEfig1
```

replaces ID fig1 for this element:

```
../concepts/c_method#mainmethod/fig1
```

### 4.3.3 Specifying a prefix for bookmarks in RTF output

Bookmarks for links in RTF output consist of an element ID with a prefix. To specify a prefix for RTF bookmarks:

```

[FileIDs]
; original map name (no extension) = prefix ID for bookmarks
mapname = prefix

```

The default bookmark prefix is the map name itself.

### 4.3.4 Checking for duplicate IDs

To be absolutely certain there are no duplicate IDs, by default **DITA2Go** conducts a duplicate-name check, and if any duplicates are found, modifies the IDs involved for uniqueness. To prevent the check for duplicates:

```

[IDOptions]
; DuplicateNameCheck = Yes (default, check for duplicate topic and
; output file names, and modify IDs for uniqueness) or No (no check)
DuplicateNameCheck=No

```

When DuplicateNameCheck=Yes, you can specify the format and length of a sequence number to be added to IDs as a suffix:

```

[IDOptions]
; UniqueNameSuffixFormat = Format string used by sprintf to add a
; unique suffix, where the int argument is a sequence number (1-999)
UniqueNameSuffixFormat = X%0.3d
; UniqueNameSuffixLength = length in bytes of the unique suffix
UniqueNameSuffixLength = 4

```

Theoretically, just the elements with duplicate IDs get suffixes. In practice, you might see many other elements with (for example) an x001 ID suffix even though no matching ID with an x002 suffix is present. That is because **DITA2Go** has to distinguish duplicates before determining whether the IDs involved are referenced. If one of them is not referenced, that ID might not be included in output.

## 4.4 Processing graphics

This section provides a brief overview of options and methods for getting the graphics referenced in your DITA 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 §40.1 [Choosing an appropriate graphics format](#) on page 745), 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.

See also:

§40 [Working with graphics](#) on page 745

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)

1. 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.
2. **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
```

3. **HTML output.** Specify the new extension:

```
[Graphics]
GraphSuffix = new_extension
```

# 5 Modifying output appearance

---

This section shows how to find and tweak format settings to get the output appearance you want. You will need the following:

- the **DITA2Go** Project Manager to run conversions
- a text editor to modify format settings
- a browser such as Firefox to view results of HTML conversions
- Microsoft Word (or another RTF-based word processor) to view results of RTF conversions.

*In this section:*

§5.1 [Understanding where to modify formats](#) on page 79

§5.2 [Understanding how to modify formats](#) on page 80

§5.3 [Changing how the output looks](#) on page 80

§5.4 [Determining how an element is rendered](#) on page 83

## 5.1 Understanding where to modify formats

The specifications that determine what your output looks like are available in format configuration files. These files contain default format definitions for nearly all DITA elements, with variations (such as indentation) for different contexts. **DITA2Go** reads these specifications to prepare CSS files that ultimately determine the appearance of HTML output, or RTF files for RTF output.

Master format configuration files are located in directory

%OMSYSHOME%\d2g\system\formats:

### Default HTML formats

d2htm\_formats.ini  
d2htm\_subformats.ini  
d2htm\_tables.ini

### Default RTF formats

d2rtf\_formats.ini  
d2rtf\_subformats.ini  
d2rtf\_tables.ini

The corresponding user-modifiable format configuration files are located in directory

%OMSYSHOME%\d2g\local\formats:

### Local HTML formats

local\_d2htm\_formats.ini  
local\_d2htm\_subformats.ini  
local\_d2htm\_tables.ini

### Local RTF formats

local\_d2rtf\_formats.ini  
local\_d2rtf\_subformats.ini  
local\_d2rtf\_tables.ini

Text-format configuration files specific to the DITA Test Suite are located in directory

%OMSYSHOME%\demo\DITATestSuite\\_config:

### DITA Test Suite HTML text formats

dtsh\_html\_formats.ini

### DITA Test Suite RTF text formats

dtsh\_rtf\_formats.ini

You will need to inspect the master files, but *do not modify them*. See §39.4.5 [Editing a format configuration file](#) on page 739.

## 5.2 Understanding how to modify formats

Make changes in appearance, such as font, style, or alignment, in local or document-specific format configuration files (never in the master format configuration files, nor in your project configuration file).

For example, in %omsyshome%\d2g\system\formats\d2htm\_formats.ini you will find the following definition of the format for top-level numbered list items:

```
[Numbered1]
based = Body
margin top = 4pt
margin bottom = 0pt
margin left = 2pt
list style = decimal
list level = 1
number = List1Num
xref = NumXref
```

If you want more space above each list item in HTML output, you would add the following setting to

%omsyshome%\d2g\local\formats\local\_d2htm\_formats.ini:

```
[Numbered1]
margin top = 12pt
```

This setting would override the space above for all HTML outputs from all DITA documents. (You must include this setting in a format configuration file; it will not take effect in your project configuration file.)

Formats can inherit properties from other formats, so this change would automatically carry through to the rest of the numbered formats, because they are based on the definition of Numbered1:

```
[Numbered1First]
based = Numbered1

[Numbered2]
based = Numbered1
```

However, this change would not affect any added paragraphs under each numbered item, because their definition overrides the space-above property:

```
[Unnumbered1]
based = Numbered1
margin top = 2pt
```

Format definitions form a tree, with inheritance, so you can adjust a particular branch, in one place. The format configuration files themselves also form trees, where the scope of a change is determined by which file you modify. See §39.4 [Deciding which configuration file to edit](#) on page 734.

## 5.3 Changing how the output looks

This section uses the DITA Test Suite project in the **DITA2Go** distribution as an example.

*In this section:*

§5.3.1 [Producing HTML output from the DITA Test Suite](#) on page 81

§5.3.2 [Getting rid of that awful green color](#) on page 81

§5.3.3 [Changing the indentation](#) on page 82

§5.3.4 [Changing the spacing](#) on page 82

### 5.3.1 Producing HTML output from the DITA Test Suite

To provide some output to work with:

1. Start the **DITA2Go** Project Manager. If you do not already have a desktop shortcut to this program, see [Step 2](#) on page 31.
2. On the **Run Options** tab, make sure both **Advanced Options** are off.
3. On the **Run Project** tab, select predefined project `DTS HTML Demo`.
4. Click **Start**, and wait for the run to complete. Now you have a collection of HTML files generated from the DITA files in the `DITATestSuite` directory.
5. In Windows Explorer, navigate to subdirectory `%OMSYSHOME%\demo\DITATestSuite\html`. You can use the HTML output files in this subdirectory to practice modifying text appearance.

### 5.3.2 Getting rid of that awful green color

Open file `design.htm` in a Web browser. You should see an HTML page with the heading **1.1 Test Suite Design**. You will modify the appearance of text on this page.

Suppose you want the paragraph after the first heading to be some color other than green. For an *ad hoc* solution, you could simply inspect the HTML to find the name of the CSS class for this paragraph, then change the color designated for that class in file `local.css`. But the next time you generated HTML output, the text would be green again.

Look at the HTML code, and notice that the CSS class for this paragraph is `shortdesc`. This is also the name of the format. To see if there is a definition of a format called `Shortdesc` for the DITA Test Suite, with a text editor inspect the DITA Test Suite format configuration file for HTML output:

```
%OMSYSHOME%\demo\DITATestSuite\_config\dts_html_formats.ini
```

(The `_config` subdirectory holds configuration files specific to the DITA Test Suite, and `dts_html_formats.ini` holds text format definitions specific to the DITA Test Suite.)

The only format definition in `dts_html_formats.ini` is this one:

```
[RevisedConref]
based = Char
color = lime
```

Therefore, we have to look further for the definition of the `Shortdesc` format. File `dts_html_formats.ini` references the following format template:

```
%OMSYSHOME%\d2g\local\formats\local_d2html_formats.ini
```

So that is the next place to look. If you are just starting out with **DITA2Go**, it is unlikely you will have put any format definitions in this user-modifiable file, so you will not likely find a definition of the `Shortdesc` format there. The next place to look is the format configuration file *referenced by* `local_d2html_formats.ini`:

```
%OMSYSHOME%\d2g\system\formats\d2html_formats.ini
```

This is the master text-format configuration file for HTML output. And sure enough, this file contains the definition:

```
[Shortdesc]
based = Body
font size = 9pt
line height = 10pt
color = green
margin left = 48pt
margin bottom = 2pt
```

You most definitely do *not* want to change the definition in the master file, because this file will be overwritten every time you update **DITA2Go**. So you must *override* the definition, in one of the “lower” files in the chain:

- `local_d2htm_formats.ini`, if you never want to see the `<shortdesc>` element rendered in green again, in any HTML output from any DITA source
- `dts_htm_formats.ini`, to get rid of the green only for the DITA Test Suite.

Suppose you choose the second option. Just copying the definition to `dts_htm_formats.ini` and deleting the `color` setting will not change the color, because the `Shortdesc` format definition inherits all properties assigned to this format in any format configuration file up the chain. So what you must do is *override* the color setting, for example, by specifying no color.

Add the following format definition to `dts_htm_formats.ini`:

```
[Shortdesc]
color =
```

You do not have to repeat any of the other properties in `dts_htm_formats.ini`, because they will be inherited from the definition at the top of the chain.

Next time you run the DITA Test Suite conversion to any HTML output, text rendered from the `<shortdesc>` element will no longer appear green.

### 5.3.3 Changing the indentation

Suppose you would also like less indentation for the `Shortdesc` format; after all, 48pt is a bit extreme. So you could add another override to the definition:

```
[Shortdesc]
margin left = 24pt
color =
```

Next time you run the DITA Test Suite conversion to any HTML output, text rendered from the `<shortdesc>` element will be indented only 24pt from the left margin.

### 5.3.4 Changing the spacing

Now suppose you want more space above and below each `Shortdesc` paragraph. The master definition of the `Shortdesc` format includes a setting for `margin bottom`, but none for `margin top`. However, the master `Shortdesc` definition inherits properties from format `Body`, on which it is based:

```
[Shortdesc]
based = Body
...
```

So, to find out what else is influencing the appearance of `Shortdesc`, you need to look at the definition of the `Body` format, which is also in `d2htm_formats.ini`:

```
[Body]
based = Para
margin top = 4pt
margin bottom = 0pt
margin left = 0pt
font size = 10pt
font family = Verdana, sans-serif
```

There you see that `margin top` is 4pt. So, to add 2pt to the space above and 2pt to the space below each `Shortdesc` paragraph, in `dts_htm_formats.ini`:

```
[Shortdesc]
margin top = 6pt
```



```
margin bottom = 4pt
margin left = 24pt
color =
```

Next time you run the DITA Test Suite conversion to any HTML output, text rendered from the `<shortdesc>` element will be set off from preceding and following text by a little more space.

To see the rest of the settings that can affect the appearance of a `Shortdesc` paragraph, look at the definition of `Body` again. `Body` is based on `Para`, so check the definition of `Para`:

```
[Para]
display = block
inline = Char
line height = 12pt
```

`Para` is the end of the line for paragraph formats; but `Para` uses character formatting from inline format `Char`, so check the definition of `Char`:

```
[Char]
display = inline
font name = Verdana, serif
font size = 10pt
```

Since `Char` is not based on any other format, now you know all the properties that combine to determine the appearance of text rendered in HTML from the `<shortdesc>` element.

See §7.6 [Configuring text output formats](#) on page 121.

## 5.4 Determining how an element is rendered

Suppose your DITA source includes element `<example>`, and you would like to know how this element will be rendered in RTF output. For this exercise, you will need to inspect four different configuration files:

```
%omsyshome%\d2g\system\config\d2g_config.ini
%omsyshome%\d2g\system\formats\d2rtf_formats.ini
%omsyshome%\d2g\system\formats\d2rtf_subformats.ini
%omsyshome%\d2g\system\lang\d2g_lang_en.ini
```

In each case, using your text editor to look for “example” will take you to the right place immediately.

To find out which output format is assigned to element `<example>`, search for “example” in the following file:

```
%omsyshome%\d2g\system\config\d2g_config.ini
```

Here you will find “example” in a section that assigns formats to elements, according to the DITA context of each element:

```
[BlockFormatMaps]
...
prereq/taskbody/*=Prereq
context/taskbody/*=Context
example/taskbody/*=Example <-----
postreq/taskbody/*=Postreq
result/taskbody/*=Result
...
```

These are all `<task>`-specific format assignments. If your `<example>` element is in a `<task>` topic, you now know that it will be rendered using output format `Example`.

If your `<example>` element is not in a `<task>` topic, you will have to add an assignment that represents the context in which the element occurs in your DITA source. In your project configuration file (`_d2rtf.ini` for RTF output) include a setting such as the following:

```
[BlockFormatMaps]
example/$body/*=Example
```

See §6.4 [Mapping element paths to output formats](#) on page 91.

(It is best to start by adding new settings to the bottom-level configuration file. Later you can move the settings to a configuration file that has a wider scope to include other projects.)

To find the definition of format `Example`, search for “Example” in the following format configuration file:

```
%omsyshome%\d2g\system\formats\d2rtf_formats.ini
```

The definition is short, because `Example` is based on format `Body`, and inherits all the properties of the latter:

```
[Example]
based = Body
runin = ExampleHead
```

This definition makes `Example` a plain body paragraph, with a run-in heading; see §6.5.4 [Assigning run-in headings to format-name prefixes](#) on page 97.

A run-in heading is a type of subformat; subformats are defined in file:

```
%omsyshome%\d2g\system\formats\d2rtf_subformats.ini
```

In this file you will not find a separate definition of the run-in heading for format `Example`, because it is identical to another run-in heading, to which it is aliased:

```
[FormatAliases]
ExampleHead = ItalicHead
```

See §7.3 [Creating aliases to existing format names](#) on page 112.

The definition of `ItalicHead` makes this heading italic:

```
[ItalicHead]
form = <i><name/> <spc/></i>
```

See §8.6 [Configuring run-in headings for text formats](#) on page 153.

Subformat component `<name/>` determines the content of the run-in heading. Content of each type of run-in heading is specified in language configuration files. For English, look in the following file:

```
%omsyshome%\d2g\system\lang\d2g_lang_en.ini
```

Here you will find text for a variety of run-in headings. For the `Example` format:

```
[RuninHeadText]
ExampleHead = For example:
```

If you want to change the content of the heading, make the change in your project configuration file. For example:

```
[RuninHeadText]
ExampleHead = [Example]
```

For internationalization, if you want headings and labels in French, you could make your project configuration file reference the French language configuration file instead:

```
[Templates]
Languages = %omsyshome%\d2g\system\lang\d2g_lang_fr.ini
```

Here the setting is a little different:

```
[RuninHeadText]
```

```
ExampleHead = Par exemple:
```



# 6 Mapping elements to output formats

---

This section shows how to map elements and their attributes to output formats. Topics include:

- §6.1 [Understanding how to assign formats](#) on page 87
- §6.2 [Specifying options for naming formats](#) on page 87
- §6.3 [Mapping outputclass attribute values to formats](#) on page 89
- §6.4 [Mapping element paths to output formats](#) on page 91
- §6.5 [Mapping element attributes to output formats](#) on page 95
- §6.6 [Specifying formats for cross references](#) on page 101
- §6.7 [Specifying formats for footnotes](#) on page 102
- §6.8 [Specifying options for figures](#) on page 102
- §6.9 [Specifying formats and options for tables](#) on page 103
- §6.10 [Specifying options for special lists](#) on page 104
- §6.11 [Specifying options for draft comments](#) on page 105
- §6.12 [Specifying options for maps](#) on page 105
- §6.13 [Deciding where to display title and shortdesc](#) on page 107

## 6.1 Understanding how to assign formats

You can assign output formats to elements based on any of the following:

- `@outputclass`; see §6.3 [Mapping outputclass attribute values to formats](#) on page 89
- element position in the DITA topic hierarchy; see §6.4 [Mapping element paths to output formats](#) on page 91
- attribute values; see §6.5 [Mapping element attributes to output formats](#) on page 95.

Failing all of these, **DITA2Go** takes the element name itself as the format name, unless you have explicitly disabled this option (see §6.2 [Specifying options for naming formats](#) on page 87). In that event, the element gets the default format for its element type; see §11.2 [Specifying properties of element types](#) on page 179.

**DITA2Go** looks for a definition of the assigned output format in the `Formats` template chain; see §7.2 [Working with format configuration files](#) on page 110:

- To define an output format, see §7 [Configuring output formats](#) on page 109.
- To specify format styling, see §8 [Configuring format components](#) on page 141.
- To change the language of predefined text used in a format, see §8.9 [Localizing output headings, labels, and names](#) on page 157.

## 6.2 Specifying options for naming formats

By default, wherever possible **DITA2Go** uses the value of the `outputclass` attribute as the name of the output format for a given element; if there is no `outputclass` attribute, **DITA2Go** uses the element mapping, and as a last resort, the element name. You can do the following:

- [Prevent outputclass as format name](#)
- [Prevent border and shading from outputclass](#)
- [Prevent element name as format name](#)
- [Specify default format names](#)

[Specify a format for <steps> headings](#)

[Specify a format for links](#)

[Specify a continuation format suffix.](#)

*Specify a  
continuation  
format*

If your DITA document includes instances of block elements nested inside other block elements, by default **DITA2Go** creates a new format for that portion of the containing block element that follows the nested block. This is because nested formats are not allowed in certain output types, notably RTF.

*Prevent  
outputclass as  
format name*

To prevent automatic use of outputclass values as output format names:

```
[ElementOptions]
; UseOutputClassForFormat = Yes (default, use whenever present) or
; No (go on to mappings in [*FormatMaps] as the next choice)
UseOutputClassForFormat = No
```

If your DITA document does not use outputclass attributes consistently, or you do not want the value of outputclass used as a format name in output, you can do either or both of the following:

- map outputclass values to other format names; see §6.3.1 [Mapping block and inline outputclass attributes to formats](#) on page 90
- map elements to formats explicitly; see §6.4 [Mapping element paths to output formats](#) on page 91.

For any element that has not been mapped to an output format either explicitly or via outputclass, by default **DITA2Go** uses the element name as the output format name.

*Prevent border  
and shading from  
outputclass*

To keep **DITA2Go** from rummaging in outputclass attributes for border and shading subformat names:

```
[ElementOptions]
; OutputclassHasBorderShadeFormats = Yes (default, look for border and
; shading format specifications in outputclass attributes), or No).
OutputclassHasBorderShadeFormats = No
```

See §8.4 [Overriding border and shading properties](#) on page 145.

*Prevent element  
name as format  
name*

To prevent automatic use of element names as output format names for unmapped elements:

```
[ElementOptions]
; UseElementNameForFormat = Yes (default, use for unmapped elements)
; or No (use defaults below if unmapped)
UseElementNameForFormat = No
```

*Specify default  
format names*

To specify default output format names for block and inline elements:

```
[ElementOptions]
; DefaultInlineFormat = format to use for unmapped inline elems
DefaultInlineFormat = Char
; DefaultBlockFormat = format to use for unmapped block elems
DefaultBlockFormat = Para
```

If you do not specify a different default, unmapped block element names become format name para in output, and unmapped inline element names become format name char in output.

*Specify a format  
for <steps>  
headings*

To specify a paragraph format name for a <steps> heading:

```
[ElementOptions]
; StepsHeadFormat = format to use for StepsHead para
StepsHeadFormat = Steps
```

The default name for the paragraph format is *Steps*.

To omit <steps> headings from output:

```
[ElementOptions]
StepsHeadFormat = Delete
```

*Specify a format for links* To specify a character format name for unmapped hypertext links that have no `outputclass` attribute:

```
[ElementOptions]
; LinkFormat = char format for unmapped hyperlinks if no outputclass
LinkFormat = Link
```

See also:

[§6.6 Specifying formats for cross references](#) on page 101

*Specify a continuation format suffix*

If your DITA source includes nested block elements, **DITA2Go** must create a new paragraph for those portions of the enclosing element that follows the enclosed elements. This is because some output types do not allow nested paragraphs. The new paragraph requires its own format, which can be derived from the format assigned to the original paragraph.

To specify a suffix to add to a format name for continued text:

```
[ElementOptions]
; ContinuedFormatSuffix = suffix to add to format name when
; a block element containing text is interrupted by another
; block element, then resumes with more text after the
; interrupting element(s), default "Cont".
ContinuedFormatSuffix = Cont
```

For example, with format `Body` defined (in system `d2htm_formats.ini`) as:

```
[Body]
based = Para
margin top = 4pt
margin bottom = 0pt
margin left = 0pt
font size = 10pt
font family = Verdana, sans-serif
```

you might want to add, in `local_d2htm_formats.ini`:

```
[BodyCont]
based = Body
margin top = 2pt
```

to adjust the spacing after a nested list. If you want the same format, unchanged, instead set:

```
[FormatAliases]
BodyCont = Body
```

Or set:

```
[ElementOptions]
ContinuedFormatSuffix =
```

to not use the suffix, just the original format name.

## 6.3 Mapping outputclass attribute values to formats

For text elements, table elements, and cross references, you can map the `outputclass` attribute of the element to an output format. However, mapping `outputclass` attributes does not apply to all elements. Among those excluded are typographic elements, such as `<b>` and `<i>`. Those are treated as properties, rather than as formats. Others excluded are `<ftn>`, `<image>`, `<alt>`, and `<index*>`. All undergo special processing of one sort or

another. For `<ftn>`, **DITA2Go** automatically uses the value of the `outputclass` attribute as the output format name.

*In this section:*

§6.3.1 [Mapping block and inline outputclass attributes to formats](#) on page 90

§6.3.2 [Mapping table outputclass attributes to formats](#) on page 90

§6.3.3 [Mapping cross-reference outputclass attributes to formats](#) on page 91

§6.3.4 [Mapping wrapper-element outputclass attributes to formats](#) on page 91

## 6.3.1 Mapping block and inline outputclass attributes to formats

To specify an output format with a name that is different from the `outputclass` attribute value of a block element (for example):

```
[BlockOutclassMaps]
; outputclass attribute of block element = paragraph format to use,
; one of the formats defined in the [Templates]Formats file.
CellBulleted = Bulleted1
```

To specify an output format with a name that is different from the `outputclass` attribute value of an inline element (for example):

```
[InlineOutclassMaps]
; outputclass attribute of inline element = character format to use,
; one of the formats defined in the [Templates]Formats file.
MarkerName = Bold
```

Assignments in these sections map existing `outputclass` attribute values to the formats you want to use. If an `outputclass` value is already the right format name, you do not need to map it.

When you explicitly assign a format to the `outputclass` for an element, that format takes precedence over the default format, and also over any format assigned in `[BlockFormatMaps]` or `[InlineFormatMaps]`; see §6.4.2 [Mapping block and inline element paths to formats](#) on page 93.

## 6.3.2 Mapping table outputclass attributes to formats

To specify a table format with a name that is different from the table `outputclass` attribute:

```
[TableOutclassMaps]
; outputclass attribute of table element = table format to use.
; one of the formats defined in the [Templates]Tables file.
FormatA = TableFormat
```

When you explicitly assign a table format to the `outputclass` for that table, that format takes precedence over the default table format. The assignments in this section map existing table `outputclass` attribute values to the table formats you want to use. You might have six different table formats in your source document, resulting in six different `outputclass` values, that should all be rendered using the same table format.

If an `outputclass` is already the right table format name, you do not need to map it. However, if you want different table formats in Word and in HTML for the same table, you would need a different `[TableOutclassMaps]` list for each output type.

To define output formats for tables, see §7.7 [Configuring table output formats](#) on page 129.

For additional ways to customize tables for specific output types, see:



§15.6 [Converting tables to print RTF](#) on page 232

§33 [Converting tables to HTML](#) on page 625.

See also:

§6.9 [Specifying formats and options for tables](#) on page 103

### 6.3.3 Mapping cross-reference outputclass attributes to formats

If the `xref` elements in your document have `outputclass` attributes, you can assign a cross-reference format name to each different `outputclass` attribute. For example:

```
[XrefOutclassMaps]
; outputclass attribute of xref element = xref format to use
Numeric = NumXref
```

See also:

§6.6 [Specifying formats for cross references](#) on page 101

### 6.3.4 Mapping wrapper-element outputclass attributes to formats

If your DITA XML document was created from material authored in a non-DITA environment such as FrameMaker, it might include `<xref>` tags automatically wrapped in `<ph>` elements, to allow `<xref>` elements in places where they would not otherwise be valid in DITA XML.

To map `<ph>` wrapper elements for cross references, index terms, and footnotes to output formats:

```
[InlineOutclassMaps]
; outputclass attribute of block or inline element = format
; For example, for the ph wrapper formats from Mif2Go:
phxref = PhXref
phindex = PhIndex
phfoot = PhFootnote
```

These format assignments are effective even if `UseOutputClassForFormat=No` (see §6.2 [Specifying options for naming formats](#) on page 87), and they override all other assignments of block and inline element formats.

For HTML output you might want to include the following settings to eliminate style tags for the wrapper elements:

```
[CharStyles]
; eliminate wrapper classes
PhXref =
PhIndex =
PhFootnote =
```

See §30.3 [Mapping character formats](#) on page 569.

## 6.4 Mapping element paths to output formats

Instead of simply mapping each element to a single output format, **DITA2Go** supports mapping *element paths*, so you can choose different output formats for the same block or inline element where it appears at different levels or in different hierarchies of your DITA document.

Most of the configuration settings that involve mapping elements to output formats use element paths as the key, to allow you to differentiate based on document hierarchy. Configuration template `d2g_config.ini` contains the full default set of maps that

**DITA2Go** uses; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

*In this section:*

- §6.4.1 [Understanding element paths](#) on page 92
- §6.4.2 [Mapping block and inline element paths to formats](#) on page 93
- §6.4.3 [Overriding element paths for default formats](#) on page 93
- §6.4.4 [Mapping the same element to different formats](#) on page 94
- §6.4.5 [Filtering out elements via format mapping](#) on page 94

## 6.4.1 Understanding element paths

When you map DITA elements to output formats, you might have to consider the position of an element in the document structure before you choose an appropriate output format. For example, the amount that text should be indented might be a function of the depth of nesting of the element that contains the text; so the same element at different levels in the hierarchy might have to be mapped to a different output format with different indentation properties. For an example, see §6.4.4 [Mapping the same element to different formats](#) on page 94.

You can restrict the mapping of a given inline or block element (or element set; see §11.1 [Defining sets of elements](#) on page 179) to a particular position in the hierarchy by specifying the element path to be considered.

An *element path* specifies the relationship of an element to the top of the top-level map. The relationship is expressed as a reversed path starting with the current element, through the root of the topic, ending with a number representing the topic depth in the current project, where 1 is the first possible level for topics. This is simply an inverted tree to the topic root, ending with the depth of the topic in the complete map (treated as though any nested maps were replaced by their top-level content).

*An element path ends with a level number*

An element path consists of the name of the current element, followed by the names of its parents in ascending order, up to and including the root XML element, and ending with a number that represents the nesting level of the root element in its XML file in the final document structure.

For example:

```
p/section/conbody/concept/3
```

This is the element path for a <p> body element in any <section> of a <concept> topic that is two levels down from the top-level topics.

When the level number of an element path is greater than 1, meaning that it is referenced from a map below the top level, **DITA2Go** generates a second path, the *extended element path*. This extended path continues up to the top level, and is normalized by omitting any `topicref/map/` pairs that are below the top level. **DITA2Go** checks the extended path first, and uses the shorter path only if the extended path has no match.

*Element path syntax*

Element names in an element path are separated by forward slashes. To specify that an element must be first under its parent, use a ^ instead of /. Wildcards ? and \* are allowed at any point.

For example:

li^ol/*	First item in any ordered list at any level
li/ol/*	Any non-first item in any ordered list at any level
li/ul/*	Any item in any unordered list at any level

## 6.4.2 Mapping block and inline element paths to formats

To map element paths of block elements to output formats (for example):

```
[BlockFormatMaps]
; block element path = paragraph format name
title/concept/1 = ChapTitle
p/conbody/concept/1 = Body
li^ol/* = NumberedFirst
li/ol/* = Numbered
li/ul/* = Bulleted
```

These settings prescribe the following formats:

- ChapTitle for every title in a concept topic that occurs at the highest topic level.
- Body for every <p> element in the <conbody> of a concept topic, at the highest topic level.
- NumberedFirst for every first <li> in <ol> anywhere in the document.
- Numbered for all other <li> elements in <ol> anywhere in the document.
- Bulleted for any <li> in a <ul> anywhere in the document.

To map element paths of inline elements to output formats (for example):

```
[InlineFormatMaps]
; inline element path = character format name
b/p/conbody/concept/1 = Bold
i/p/conbody/concept/1 = Italic
```

These settings indicate that <b> elements should be mapped to a format named *Bold* wherever those elements occur in the DITA document; and that <i> elements should be mapped to a format named *Italic*, wherever they occur.

**Note:** If you explicitly assign a format to the outputclass for an element in [BlockOutclassMaps] or [InlineOutclassMaps], that format takes precedence over any format assigned to the element in [BlockFormatMaps] or [InlineFormatMaps]. See §6.2 [Specifying options for naming formats](#) on page 87.

Configuration template `d2g_config.ini`, located in `%omsyshome%\d2g\configs`, includes elephant paths for all default formats included in the **DITA2Go** distribution.

## 6.4.3 Overriding element paths for default formats

DITA maps and projects can be set up many different ways. The default output formats (and their mappings from elements) included in the **DITA2Go** distribution reflect only one possible configuration. You can find and inspect the default element paths in distribution file `%omsyshome%\d2g\configs\system\d2g_config.ini`.

For example, for glossary content for Help systems, **DITA2Go** covers the bookmap case. In `d2g_config.ini`:

```
[BlockFormatMaps]
title/$topic/glossarylist/booklists/backmatter/$map/1=GlossaryTitle
glossterm/* = Heading1
glossdef/* = Abstract
```

And in `d2help_config`, *Heading1* is one level under *GlossaryTitle*:

```
[HelpContentsLevels]
GlossaryTitle=2
Heading1*=3
```

so the glossary terms nest under the glossary title.

However, if you have the glossary nested under a top-level topic, the glossary terms are at the same level. The fix is to override the default mapping and lower the glossary terms one level:

```
[BlockFormatMaps]
glossterm/* = Heading2
```

That way the glossary terms nest under the title topic.

#### 6.4.4 Mapping the same element to different formats

Suppose you want to eliminate an indent from the format for a given block element when that element occurs within a table. For example, when a simple list is nested in a <table> cell or even a <simpletable> cell, you might want to override the indent for the SimpleListItem format. This format is assigned by default to <sli> via an element path defined in file %OMSYSHOME%\d2g\system\config\d2g\_config.ini:

```
[BlockFormatMaps]
sli/sl/* = SimpleListItem
```

For HTML output, format SimpleListItem is defined in %OMSYSHOME%\d2g\system\formats\d2htm\_formats.ini:

```
[SimpleListItem]
based = Body
margin top = 2pt
margin bottom = 0pt
margin left = 12pt
list style = none
list level = 1
```

In %OMSYSHOME%\d2g\local\formats\local\_d2htm\_formats.ini you can define an alternate format, based on [SimpleListItem]:

```
[SimpleTableListItem]
based = SimpleListItem
margin left = 0pt
```

Then in your project configuration file you can include alternate element paths for <sli>:

```
[BlockFormatMaps]
; This mapping handles a simple list in a table cell:
sli/sl/entry/* = SimpleTableListItem
; This mapping handles a simple list in a simpletable cell:
sli/sl/stentry/* = SimpleTableListItem
```

Depending on the complexity of nested lists in your document, you might need to provide additional override formats for alternate element paths that start with <sli>.

#### 6.4.5 Filtering out elements via format mapping

As an adjunct or alternative to ditaval filtering (see §9 [Specifying conditional processing](#) on page 161), you can exclude specific elements from output by mapping them to format property Delete. For example:

```
[BlockFormatMaps]
context/* = Delete
stepresult/* = Delete
info/* = Delete

[InlineFormatMaps]
draft-comment/* = Delete
```

## 6.5 Mapping element attributes to output formats

For some kinds of elements, such as `<note>`, you might want to vary the output format based on one or more attributes other than the `outputclass` attribute (see §6.3 [Mapping outputclass attribute values to formats](#) on page 89). You can specify that the values of certain attributes should modify the output format that **DITA2Go** uses for a block or inline element, and you can also specify a run-in heading to be included on output.

*In this section:*

§6.5.1 [Listing elements whose attributes can affect output formats](#) on page 95

§6.5.2 [Listing attributes whose values can affect output formats](#) on page 95

§6.5.3 [Assigning format-name prefixes to attribute values](#) on page 96

§6.5.4 [Assigning run-in headings to format-name prefixes](#) on page 97

§6.5.5 [Deciding which formats need a run-in heading property](#) on page 99

§6.5.6 [Understanding the order of prefixes for multiple attributes](#) on page 100

§6.5.7 [Understanding how prefixes modify output formats](#) on page 100

§6.5.8 [Understanding default attribute-based prefixes and headings](#) on page 100

### 6.5.1 Listing elements whose attributes can affect output formats

To specify that **DITA2Go** should apply a prefix to the name of the output format assigned to an element, based on one or more attributes of that element:

```
[ElementAttrPrefixes]
; element name = section with list of attrs for which
; prefixes should possibly be applied to the format name
element = ElemSectionName
```

By default:

```
[ElementAttrPrefixes]
note = NoteAttrPrefixes
step = StepAttrPrefixes
* = AttributePrefixes
```

The element name can include wildcards. These default settings are located in configuration template `%omsyshome%\d2g\system\config\d2g_config.ini`; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

The setting for `<note>` directs **DITA2Go** to look up the attributes listed in configuration section `[NoteAttrPrefixes]`, and to use those attributes to determine which *Note* format to apply to a given instance of `<note>`; or, in effect, how to modify the base `[Note]` format. And similarly for `<step>`.

The last setting above directs **DITA2Go** to look up the attributes listed in configuration section `[AttributePrefixes]` for *all* elements. (The `importance` attribute is included there by default; see §6.5.2 [Listing attributes whose values can affect output formats](#) on page 95.)

### 6.5.2 Listing attributes whose values can affect output formats

To list the attributes that **DITA2Go** should use to add format-name prefixes for a given element, in the section assigned to that element in `[ElementAttrPrefixes]` (see §6.5.1 [Listing elements whose attributes can affect output formats](#) on page 95):

```
[ElemSectionName]
; attribute name = section with list of values, or
; attribute name = * to use the value itself as a prefix
```

```
attributename = AttrSectionName
```

The attribute name cannot include wildcards.

When *attributename*=\*, **DITA2Go** prefixes the format name for the element with the value of the attribute.

When *attributename*=*AttrSectionName*, **DITA2Go** looks up the attribute in [*AttrSectionName*] to determine what prefix to apply.

By default:

```
[NoteAttrPrefixes]
type = *

[StepAttrPrefixes]
importance = StepImportancePrefixes

[AttributePrefixes]
importance = ElemImportancePrefixes
```

These default settings are located in configuration template

%omshome%\d2g\system\config\d2g\_config.ini; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

In the first setting, for each <note> element, **DITA2Go** prefixes the name of the Note format with the value of the *type* attribute.

In the second setting, for each <step> element, **DITA2Go** looks up the value of the *importance* attribute in configuration section [*StepImportancePrefixes*], to determine what prefix to use for that instance of <step>. See §6.5.3 [Assigning format-name prefixes to attribute values](#) on page 96.

In the third setting, for *every* element that includes an *importance* attribute, **DITA2Go** looks up its value in configuration section [*ElemImportancePrefixes*], to determine what prefix to use for the format name.

### 6.5.3 Assigning format-name prefixes to attribute values

To specify what to do with each value of each attribute to which *AttrSectionName* is assigned in [*ElemSectionName*] (see §6.5.2 [Listing attributes whose values can affect output formats](#) on page 95):

```
[AttrSectionName]
; attribute value = prefix
value1 = prefix1
value2 = prefix2
. . .
```

The attribute value can include wildcards. **DITA2Go** considers attribute values in the order they are listed in [*AttrSectionName*]; the first attribute value in the list that matches the value for a given instance of an element determines the prefix applied to the format name for that instance of that element. If the prefix assigned to a value is “\*”, the attribute value itself becomes the prefix. If the prefix is blank, no prefix is applied for that value.

By default:

```
[StepImportancePrefixes]
required = ReqStep
optional = OptStep

[ImportancePrefixes]
required = Req
```

```
optional = Opt
* = *
```

These default settings for required and optional values of the step element importance value are located in configuration template

%omsyshome%\d2g\system\config\d2g\_config.ini; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

For example:

```
[NoteTypePrefixes]
note =
warning = warn
* = *
```

In this last example, a given <note> element would be mapped to a format name prefixed according to the value of the type element:

- If @type=note, **DITA2Go** would not apply a prefix to the format name.
- If @type=warning, the prefix warn would be applied to the format name; for example, warnNote.
- For any other value of the type attribute, **DITA2Go** would apply the attribute value itself as a prefix to the format name; for example, cautionNote.

## 6.5.4 Assigning run-in headings to format-name prefixes

For block and inline elements, you can use the prefixes that are to be tacked onto format names to specify the text and format of a run-in heading that will precede the content of the element itself on output. For example:

**Required:** Churzen the framble with the supplied grappitz.

In this example, **Required:** is a run-in heading, with a character format different from that of the content.

**DITA2Go** supplies default formats and text for <note> and <step> run-in headings. These default formats are included in the following subformat configuration files:

```
%omsyshome%\d2g\system\formats\d2*_subformats.ini
```

The text for the default run-in headings comes from a language configuration file:

```
%omsyshome%\d2g\system\lang\d2g_lang_en.ini
```

In other words, the text is not taken from the attribute value; the latter only determines the name of the output format for the element in question.

To associate a run-in heading with a format-name prefix:

[Assign a run-in heading format name to the prefix](#)

[Define the run-in heading format](#)

[Provide content for the run-in heading.](#)

*Assign a run-in heading format name to the prefix*

To assign the name of a character format to a run-in heading for a prefix:

```
[BlockFormatPrefixRunins]
; prefix = default runin format to be used if no prefixed
; format exists and if base format has no runin itself
prefix = PrefixFormatHead
```

For example:

```
[BlockFormatPrefixRunins]
ReqStep = ReqStepHead
OptStep = OptStepHead
Req = RequiredHead
```



```
Opt = OptionalHead
warning = WarningHead
```

You can also assign a character format to a run-in heading for a prefix to the content of an inline element:

```
[InlineFormatPrefixRunins]
; prefix = default runin format to be used if no prefixed
; format exists and if base format has no runin itself
prefix = PrefixFormatHead
```

Default assignments of run-in headings to prefixes are located in the following configuration file:

```
%omsyshome%\d2g\system\config\d2g_config.ini
```

See §39.1.3 [Understanding how templates are chained together](#) on page 728.

*Define the run-in  
heading format*

To specify the style for a run-in heading:

```
[PrefixHead]
form = format components
```

For example:

```
[ReqStepHead]
form = <b>[<name/>] </b>

[OptStepHead]
form = <i>[<name/>] </i>

[RequiredHead]
form = <b><name/>: </b>

[OptionalHead]
form = <i><name/>: </i>

[WarningHead]
form = <b><name/>!</b><br/>
```

Define run-in heading formats in a Subformats configuration file, see §8.6 [Configuring run-in headings for text formats](#) on page 153.

*Provide content  
for the run-in  
heading*

In place of the <name/> building block, you can include fixed text for a run-in heading. For example:

```
[DangerHead]
form = <b><i>!! DANGER !!</i></b>\\n
```

However, the <name/> building block allows you to specify alternate text in different languages.

To specify text for the <name/> building block of a run-in heading, in a language configuration file:

```
[RuninHeadText]
; used as the <name/> content in the matching subformat
PrefixHead = Text of heading
```

To cancel run-in headings individually, set them to nothing (empty).

Run-in headings for notes are based on the type attribute of the <note> element:

```
[RuninHeadText]
NoteHead = Note:
AttentionHead = Attention:
CautionHead = Caution:
WarningHead = Warning:
DangerHead = Danger:
FastpathHead = Fastpath:
ImportantHead = Important:
```



```
RememberHead = Remember:
RestrictionHead = Restriction:
TipHead = Tip:
OtherHead = Note:
```

Importance run-in headings are based on the importance attribute value:

```
[RuninHeadText]
optional = Optional:
required = Required:
urgent = Urgent!
obsolete = [obsolete]
deprecated = [deprecated]
default =
low =
medium =
high =
recommended =
```

Run-in headings for task elements:

```
[RuninHeadText]
; ReqStepHead = at start of step with importance=required
ReqStepHead = [Required step]
; OptStepHead = at start of step with importance=optional
OptStepHead = [Optional step]
; StepsHead = at start of steps, in its own para with StepsFormat
StepsHead = Procedure:
; PrereqHead = at start of prereq element
PrereqHead = Before you start:
; ContextHead = at start of context element
ContextHead = Purpose:
; ExampleHead = at start of example element
ExampleHead = For example:
; ResultHead = at start of result element
ResultHead = Results:
; PostreqHead = at start of postreq element
PostreqHead = After you finish:
; StepxmpHead = at start of stepxmp element
StepxmpHead = Example:
; StepresultHead = at start of stepresult element
StepresultHead = Result:
```

The ExampleHead text also applies to <example> in other topic types. StepsHead is a separate paragraph.

Specify run-in heading content for the <name/> building block in a Languages configuration file; see §8.9 [Localizing output headings, labels, and names](#) on page 157.

The default settings above are in the following file:

```
%omsyshome%\d2g\system\lang\d2g_lang_en.ini
```

### 6.5.5 Deciding which formats need a run-in heading property

Instead of considering only the current element to determine prefixes and formatting, **DITA2Go** also looks up one level in the current topic in your DITA document. If the parent element does not have text of its own, and if the current element is the first child of that parent, **DITA2Go** processes the parent to determine what its format would be if it did have text. This process also identifies any applicable prefix. Then **DITA2Go** processes the current element, and determines its format; however, a prefix has already been identified from the parent. That parent prefix becomes the first prefix for the current element.

For example, if your document uses the `<note>` element as a wrapper for other text elements, `<note>` itself might have no text content, and therefore **DITA2Go** would not automatically look for an output format for `<note>`. However, if another text element is nested in `<note>`, **DITA2Go** checks `<note>` for possible format prefixes to apply to the nested element.

### 6.5.6 Understanding the order of prefixes for multiple attributes

If you specify more than one attribute whose values should be considered for prefixing the format name for a given element `<elem>`, the order in which you list the attributes to be considered in `[ElemSectionName]` (see §6.5.2 [Listing attributes whose values can affect output formats](#) on page 95) determines the order in which the prefixes will be applied:

```
[ElemAttrPrefixes]
attrx = *
attry = *
```

If the default format name for `<elem>` is `Elem`, for each instance of `<elem>` that includes values for both `@attrx` and `@attry`, their values will be prefixed to the format name in the order the attributes are listed in this section (not the order they are listed in the element itself), and the name of the format **DITA2Go** looks for would be `xvalueyvalueElem`.

### 6.5.7 Understanding how prefixes modify output formats

You do not have to define a different format for every possible combination of attribute values for which you specify format-name prefixes, unless you want an effect different from what you would get when content rendered using the base format is prefixed by any run-in heading(s) assigned to the prefix(es). Most often you will not have to define any prefixed formats at all; the presence of a candidate attribute effectively includes the assigned run-in heading in the output for an instance of a given element.

If you have not defined formats named for all the candidate attribute-value combinations for elements listed in `[ElementAttrPrefixes]` (see §6.5.1 [Listing elements whose attributes can affect output formats](#) on page 95), **DITA2Go** uses the base format name, and modifies the output by prefixing content with the run-in headings (if any) assigned to the candidate attribute values for that instance of the element (see §6.5.4 [Assigning run-in headings to format-name prefixes](#) on page 97). If the base format itself includes the `runin` property (see §7.6.2 [Specifying default properties for character formats](#) on page 121 and §7.6.3 [Specifying default properties for paragraph formats](#) on page 122), its run-in heading follows the attribute run-in heading(s).

If a base format for the element is neither defined nor listed in `[FormatAliases]` (see §7.3.2 [Mapping legacy names to defined formats](#) on page 113), the output for that element will have only the default format properties for block or inline elements.

### 6.5.8 Understanding default attribute-based prefixes and headings

Your **DITA2Go** distribution includes default attribute-based format-name prefixes for `<note>` elements, and default prefixes based on the `importance` attribute for all elements, in system configuration file `d2g_config.ini`. If you want run-in headings different from those determined by these attributes, or no run-in headings at all, you must override the default configuration settings with your own.

The default for a `<note>` element is to use a run-in format (in system format configuration file `d2htm_subformats.ini`), which has a `<name/>` component. The actual name

comes from system language configuration file `d2g_lang_en.ini`. The name does *not* come from the attribute value.

Type attribute for  
<note> elements

For <note>, the value of the `type` attribute (if any) determines which run-in heading will be prefixed to the text. The name of the format of the run-in heading consists of the value of the `type` attribute prefixed to `Head`. The content of the run-in heading consists of the text assigned to the name of the run-in heading format, in the `[RuninHeadText]` section of whichever language configuration template is in force. If a particular <note> element has no `type` attribute, **DITA2Go** does not include a run-in heading for that instance, unless you specify a default value for `@type`.

To specify a default type for any <note> element that lacks a `type` attribute; for example, note:

```
[ElementOptions]
; DefaultNoteType = value to use for @type if it is omitted;
; default is nothing, per the DITA spec.
DefaultNoteType = note
```

If you do not specify a value for `DefaultNoteType`, no run-in heading is included in the output for any instance of <note> that does not contain a `type` attribute.

**Note:** The DITA Open Toolkit assumes the value `note` when `@type` is missing.

Importance  
attribute for all  
text elements

For all text elements, the value of the `importance` attribute (if any) determines a format-name prefix, and the content of the associated run-in heading is specified by the run-in heading format assigned to each importance value. For <step>, importance attribute values `required` and `optional` specify a special run-in heading for the `Step` format.

## 6.6 Specifying formats for cross references

Cross references (empty `xref` elements) need different output formats depending on the type of element referenced. Numbered elements, such as list items and footnotes, need the number included in cross references to those elements; headings typically need the title included in the cross reference.

If the <xref> elements in your document include an `outputclass` attribute, you can map `outputclass` values to formats; see §6.3.3 [Mapping cross-reference outputclass attributes to formats](#) on page 91.

If there is no `outputclass` attribute, **DITA2Go** uses the `xref` property of the element to which the cross reference refers (see §7.6.6 [Specifying block properties for paragraph formats](#) on page 124). Failing that, **DITA2Go** assigns a default cross-reference format based on the `xref` element `type` attribute:

```
NumXref      for li (default format = <numonly/>)
FtnXref      for fn (default format = <sup><numonly/>)
TextXref     for other (default format = <paratext/>)
TitleXref    for the rest (default format= <autonum/> <paratext/>)
```

To specify a cross-reference format name for each type of cross reference:

```
[ElementOptions]
; XrefTextFormat = name for xrefs to text elements
XrefTextFormat = TextXref
; XrefTitleFormat = name for xrefs to elements with titles
XrefTitleFormat = TitleXref
; XrefNumFormat = name for xrefs to li elements
XrefNumFormat = NumXref
; XrefFtnFormat = name for xrefs to footnotes,
```

```
XrefFtnFormat = FtnXref
; GenListXrefFormat = name for xrefs used in generated list items
GenListXrefFormat = TitleXref
```

These are the default names for cross-reference formats. You can specify different names; however, if you do, you must change the default names wherever they occur, notably in subformats configuration templates; see §8.1.1 [Understanding where to define format components](#) on page 141.

To configure the formats for each of these cross-reference types, see §8.7 [Defining cross-reference output formats](#) on page 155.

## 6.7 Specifying formats for footnotes

To specify the names of output formats for footnotes in text and in tables:

```
[Footnotes]
; FootnoteFormat = format to use for footnote elements in text
FootnoteFormat = Footnote
; TblFootFormat = format to use for footnote elements in tables
TblFootFormat = TblFootnote
```

These are the default names for footnote formats. If you change the names, you must change them wherever they are referenced, notably in format configuration templates; see §39.1.5 [Understanding how format templates are organized](#) on page 730.

To define formats for footnote text, see §7.6 [Configuring text output formats](#) on page 121.

To define formats for footnote references, see §8.5.4 [Defining footnote numbering](#) on page 150.

**Note:** Footnotes with IDs use cross references only.

## 6.8 Specifying options for figures

You can specify options that affect title placement, use of anchors, some aspects of alignment, and numbering of figures.

### *Title placement*

By default, **DITA2Go** puts figure titles below images on output, but puts the IDs for figure titles above the images, so that links to an image in HTML show the whole image and the title, not just the title.

To keep the figure title above the image:

```
[FigureOptions]
; FigTitleAboveImage = No (default, links moved so browsers show the
; figure, not just the title), or Yes (DITA style, title above image)
FigTitleAboveImage = Yes
```

If you use a `fig` element to hold a `simplet` (so the `simplet` can have a title; see §14.6 [Treating figure titles as table titles](#) on page 203), by default **DITA2Go** puts the title above the table on output, which generally looks better.

To keep the figure title below the table:

```
[FigureOptions]
; FigTitleAboveTable = Yes (default, figure titles look better above
; a table) or No (normal figure style, title below table)
FigTitleAboveTable = No
```

### *Anchor*

Some output types might require figures to be anchored in paragraphs.

To specify a format for figure-anchor paragraphs:

```
[FigureOptions]
; UseFigureAnchor = Yes (default, use anchor para) or No (for special
; cases, will break many normal use cases)
UseFigureAnchor = Yes
; FigureAnchorFormat = name of format to use for figure anchor para
; when TitleAboveImage = No
FigureAnchorFormat = FigurePara
```

Anchor paragraphs in table cells should contain at least one character in addition to the image, even if the character is just a nonbreaking space, to keep the image inline.

**Alignment** To specify alignment on output for any image with attribute @placement="break":

```
[FigureOptions]
; GraphicAlignment = image alignment when placement="break": left,
; right, or center, default left
GraphicAlignment = left
```

**Note:** **DITA2Go** ignores all special <fig> attributes except for @placement, @width, @height, and @alt.

**Numbering** To specify numbering and a label for figure titles, see §8.5.3 [Defining number format components](#) on page 148.

## 6.9 Specifying formats and options for tables

To specify a default format for each DITA table type:

```
[TableOptions]
; NormalTableFormat = format for normal tables without outputclass
NormalTableFormat = NormalTable
; SimpleTableFormat = format for simpletables without outputclass
SimpleTableFormat = SimpleTable
; ChoiceTableFormat = format for choicetables without outputclass
ChoiceTableFormat = ChoiceTable
; PropertiesTableFormat = format for properties without outputclass
PropertiesTableFormat = PropTable
```

To specify a different format for selected tables, you must give those tables an outputclass attribute; see §6.3.2 [Mapping table outputclass attributes to formats](#) on page 90.

If a table in your DITA document already has an outputclass attribute, and a table format of the same name is defined in your project configuration file or in a Tables configuration file, **DITA2Go** uses that format for the table in question, instead of the default format.

To specify a format for a paragraph used in output to contain a table:

```
[TableOptions]
; UseTableAnchor = Yes (default, use anchor para) or No
UseTableAnchor=Yes
; TableAnchorFormat = name of para fmt used to contain tables
TableAnchorFormat = TableAnchorPara
```

The default value of TableAnchorFormat is TableAnchor; this format is defined in the formats configuration files in %OMSYSHOME%\d2g\system\formats.

For selected table formats you can override use of the anchor format named by TableAnchorFormat:

```
[TableAnchorFormats]
ThisTableFormat = OtherAnchorPara
```

To assign an @outputclass to table footer rows:

```
[TableOptions]
; TableFooterClass = @outputclass used in <row> to identify a row
; that should be processed as a table footer; case-insensitive.
TableFooterClass = FooterClass
```

The default value of TableFooterClass is footer.

Although DITA tables do not include the notion of a footer row, you can designate <row> elements to be treated as footer rows on output. Any <row> element that contains an outputclass attribute with the value assigned to TableFooterClass will get the format defined for that value, provided UseOutputClassForFormat=Yes; see §6.2 [Specifying options for naming formats](#) on page 87.

To define output formats for tables, see §7.7 [Configuring table output formats](#) on page 129.

For additional ways to customize tables for specific output types, see:

§15.6 [Converting tables to print RTF](#) on page 232

§33 [Converting tables to HTML](#) on page 625.

## 6.10 Specifying options for special lists

By default, **DITA2Go** renders definition lists and parameter lists as tables rather than as lists. You can have them rendered as text lists instead, except when they occur inside tables. This means you can, for example, use a list format for ePub output and a table format for HTML or PDF output, without involving conditions or making any change in the source topics or maps.

To render definition or parameter lists as lists, not tables:

```
[TableOptions]
; DefinitionListTables = Yes (default, render definition lists as
; tables) or No (render them as text lists).
DefinitionListTables = No
; ParameterListTables = Yes (default, render parameter lists as
; tables) or No (render them as text lists).
ParameterListTables = No
```

To specify table formats for definition lists and parameter lists when they are rendered as tables:

```
[TableOptions]
; DefListTableFormat = table format when DefinitionListTables=Yes
DefListTableFormat = DefListTable
; ParamListTableFormat = table format when ParameterListTables=Yes
ParamListTableFormat = ParamListTable
```

These are the default table formats for definition lists and parameter lists; you can find them in %OMSYSHOME%\d2g\system\formats\d2htm\_tables.ini and %OMSYSHOME%\d2g\system\formats\d2rtf\_tables.ini. Copy the format definitions to the corresponding local format configuration files to modify them.

**Note:** Formats for list *items* that occur in tables converted from special lists always have names that end in Tbl.

To override the default relative column widths (25% and 75%) for definition and parameter lists when they are rendered as tables:

```
[TableOptions]
DefListTableColWidths = leftcolwidth rightcolwidth
ParamListTableColWidths = leftcolwidth rightcolwidth
```

These settings take the standard DITA column width syntax, overriding the built-in 25\*75\* setting.

## 6.11 Specifying options for draft comments

By default, **DITA2Go** includes the content of `<draft-comment>` elements in output.

To exclude `<draft-comment>` content from output:

```
[ElementOptions]
; KeepDraftComments = Yes (default, include draft-comment content
; in output) or No (exclude draft-comment from output)
KeepDraftComments = No
```

When `KeepDraftComments=Yes` (the default), **DITA2Go** treats `<draft-comment>` as an inline element; however, it is one of those elements that can contain “pernicious mixed content”: both block and inline elements, and also text.

By default, **DITA2Go** maps `<draft-comment>` content to a format that makes it easy to spot in the output:

```
[InlineFormatMaps]
draft-comment/* = DraftComment
```

And in system format configuration files, **DITA2Go** defines this format as follows:

```
[DraftComment]
display = inline
color = red
```

You can override this definition in a local format configuration file.

However, if there are any other block or inline formats within `<draft-comment>`, they will override its format, unless you assign a format for each; for example:

```
[BlockFormatMaps]
p/draft-comment/* = DraftPara
```

and:

```
[DraftPara]
based = Body
color = red
```

See §7 [Configuring output formats](#) on page 109.

## 6.12 Specifying options for maps

*In this section:*

- §6.12.1 [Providing default output formats for map content](#) on page 105
- §6.12.2 [Including shortdesc content in the title attribute](#) on page 106
- §6.12.3 [Including title-only topics in output](#) on page 106
- §6.12.4 [Including children of topic headings](#) on page 107

### 6.12.1 Providing default output formats for map content

To specify a base name to use in default format names for ditamap and bookmap elements:

```
[ElementOptions]
; BaseMapFormat = base format name to use for ditamap elements
BaseMapFormat = Map
```



**DITA2Go** appends a two-digit document level number to the base format name for each map level, such as Map02. The default base name is Map.

**DITA2Go** treats the value of `BaseMapFormat` as the default format name for map elements for which the `[BlockFormatMaps]` section does not contain a contextual mapping. The main use of `BaseMapFormat` is to help identify such omissions, so you can map those elements to new or existing formats. An exhaustive `[BlockFormatMaps]` section would be very large, and pretty unmanageable.

*See also:*

§14.3 [Generating a table of contents](#) on page 198

## 6.12.2 Including shortdesc content in the title attribute

To include `shortdesc` content in the `title` attribute of the heading it applies to, instead of as a paragraph following the heading:

```
[MapOptions]
; UseMapDescAsTitle = No (default, treat descriptions as text if used)
; or Yes (use descriptions without formatting as title attributes)
UseMapDescAsTitle = Yes
```

When `UseMapDescAsTitle=Yes`, the `shortdesc` text appears with a mouseover in HTML output. However, for the topic text itself, the `shortdesc` always appears after the title, unless you map it to one of the following:

- `[BlockFormatMaps]None` (see §6.4.2 [Mapping block and inline element paths to formats](#) on page 93)
- `[HTMLParaStyles]Delete` (see §30.2.6 [Eliminating unwanted paragraphs](#) on page 569)
- `[WordStyles]Delete` (see §15.4.4 [Omitting content from RTF output](#) on page 228).

*See also:*

§6.13 [Deciding where to display title and shortdesc](#) on page 107

§13.4 [Including descriptions with related links](#) on page 191

§22.9 [Providing hover text for links in HTML](#) on page 441

## 6.12.3 Including title-only topics in output

By default, **DITA2Go** treats all `<topichead>` elements as title-only topics, so they behave the same in generating content (their `navtitle`) in both HTML and print (RTF) output. This is also true for other topic heads derived from `<topicref>` that do not have an `href` attribute, where the title is the `<navtitle>` and the `<shortdesc>` element is in `<topicmeta>`.

To turn this behavior off and include such topic titles only in the TOC:

```
[TopicHeads]
; TreatTopicheadsAsTopics = Yes (default, treat topicheads as topics
; or No (include them in the TOC only).
TreatTopicheadsAsTopics = No
```

To specify a root element and ID base name for title-only topics:

```
[TopicHeads]
; TitleOnlyTopicType = root element to use for virtual topics that are
; generated from topicheads, any topic root; default "topic"
TitleOnlyTopicType = topic
; TitleOnlyTopicID = base name for @id for virtual generated topics,
```



```
; to which a sequence number 1..n is added; default "head"
TitleOnlyTopicID = head
```

## 6.12.4 Including children of topic headings

If your document has a bookmap that includes part elements, by default their titles appear in output only with related topics, mainly siblings. You can direct **DITA2Go** to include the titles of their child topics as well.

To include titles of topic-heading children:

```
[TopicHeads]
; AddTopicHeadChildren = No (default) or Yes
AddTopicHeadChildren = Yes
; TopicHeadChildHeadFormat = format for topic head children heading;
; the children themselves use the [RelatedLinks]TopicTitleFormat.
TopicHeadChildHeadFormat = TopicHeadChildHead
```

When `AddTopicHeadChildren=Yes`, **DITA2Go** adds related child links to topic-heading topic content, provided the following is true:

```
[RelatedLinks]
GenerateParentChild = Yes
```

Setting `AddTopicHeadChildren=Yes` overrides the value of `UseChildren` (see §13.2 [Generating and including related links](#) on page 189). The added items use the same formats as other related links.

To specify the text of an introductory heading for a list of child topics, in a language configuration file:

```
[TopicHeadText]
TopicHeadChildHead = In this section:
```

See §8.9 [Localizing output headings, labels, and names](#) on page 157. The text shown here is the default value specified in `d2g_lang_en.ini`, located in directory `%OMSYSHOME%\d2g\system\lang`.

See also:

§13.2 [Generating and including related links](#) on page 189

## 6.13 Deciding where to display title and shortdesc

For Help output, the TOC that appears in the tripane is generated by a process that does not permit adding `shortdesc` text. Otherwise, text of `title` and `shortdesc` elements can appear in three different places:

1. Always at the beginning of the topic itself (unless you deliberately skip the `shortdesc` element by mapping it out).
2. In the TOC (but only for HTML, XHTML, and RTF outputs in the case of `shortdesc`); see §14.3 [Generating a table of contents](#) on page 198.
3. In related links to the topic, which appear at the bottom of the referencing topics, but only if `[RelatedLinks]UseRelDescription=Yes`; the default is `No`. See §13.4 [Including descriptions with related links](#) on page 191.

In places 2 and 3 (but not 1), if you have chosen to use `shortdesc`, you have a further choice to make. The default is to have `shortdesc` appear as text in a paragraph after the text of the `title` element. You can change that:

- for the TOC, with `UseMapDescAsTitle=Yes`; see §6.12.2 [Including shortdesc content in the title attribute](#) on page 106

- for related links, with `UseRelDescAsTitle=Yes`; see §13.4 [Including descriptions with related links](#) on page 191

in which case `shortdesc` does *not* appear as text, but displays when you mouse over the title (in HTML output); this is because it is instead in the `title` attribute of the title in the TOC or the related link.

**DITA2Go** uses the map `shortdesc` in the links if possible, otherwise the `shortdesc` in the `reltable`, or the `linktext` or `desc` in the `related-links` section, or finally the `shortdesc` of the referenced topic itself.

# 7 Configuring output formats

---

This section shows how to specify output styles and display characteristics for all paragraph, character, and table output formats assigned to elements in your DITA document, as well as to page layouts for RTF output. Topics include:

- §7.1 [Understanding the purpose of output formats](#) on page 109
- §7.2 [Working with format configuration files](#) on page 110
- §7.3 [Creating aliases to existing format names](#) on page 112
- §7.4 [Understanding how to define output formats](#) on page 114
- §7.5 [Understanding text output formats](#) on page 119
- §7.6 [Configuring text output formats](#) on page 121
- §7.7 [Configuring table output formats](#) on page 129
- §7.8 [Configuring page layouts for RTF output](#) on page 134
- §7.9 [Inserting line, column, and page breaks in output](#) on page 138

## 7.1 Understanding the purpose of output formats

**DITA2Go** uses output formats, or styles, as a “presentational vocabulary” for the DITA elements that form the “semantic vocabulary” of your document. Using formats as identifiers is a key part of **DITA2Go** architecture. Mechanisms for combining element names, element context, outputclass, PI markers, and configuration settings are already in place for formats. A format is more descriptive than any of its sources.

Output formats provide the “glue” between your DITA elements and the resulting output. Many settings in your project configuration file consist of a format name assigned to a particular element or group of elements; subsequent settings assign actions or consequences to those format names. Output format definitions assign style properties to those format names, to determine how content will look.

To get you started, **DITA2Go** provides format configuration templates that define a set of default output formats and specify their style properties; see §39.1.5 [Understanding how format templates are organized](#) on page 730. Your project configuration file references these through a chain of configuration templates; see §39.2 [Referencing configuration files and templates](#) on page 731. The default format names are those shown in §6 [Mapping elements to output formats](#) on page 87.

Format definitions make understanding a particular format simpler, remove the possibility of misspelling the format name in one of many places, and make it possible to clone a format to make a variant with just a single copy/paste action.

You can specify properties for each format assigned to an element in your DITA document, in a format configuration file. **DITA2Go** uses the specifications in format configuration files to generate a Word stylesheet, CSS for HTML, or whatever other stylesheet is required by the output type. If you include a setting that is not used by the current output type, **DITA2Go** ignores that setting. Therefore you can have both RTF-specific and HTML-specific settings in the same format configuration section, and only the appropriate settings will be used. That should make coordinating formats for different outputs fairly easy.

## 7.2 Working with format configuration files

Specifications for output formats must be included in separate format configuration files, not in your project configuration file.

*In this section:*

- §7.2.1 [Understanding where to define output formats](#) on page 110
- §7.2.2 [Specifying paths to your own format configuration files](#) on page 110
- §7.2.3 [Understanding how DITA2Go builds format chains](#) on page 111
- §7.2.4 [Understanding how DITA2Go processes format chains](#) on page 112

### 7.2.1 Understanding where to define output formats

Your **DITA2Go** distribution includes a full set of format configuration templates and editable format configuration files, organized in side chains referenced from the general configuration chain; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

You can override the default format definitions, and define additional formats, in the provided editable format configuration files paired with the templates. Or, you can set up your own configuration template chain for format definitions, and reference that chain from your project configuration file. See §7.2.3 [Understanding how DITA2Go builds format chains](#) on page 111.

Your project configuration file can reference templates for any or all of the following kinds of formats:

- text formats (paragraph and character)
- table formats
- format components: number streams, cross references, borders, and shading
- page layouts for RTF output.

The text-format template is the main template in this collection; it can reference the other templates, but not vice versa. See §39.5.3 [Chaining configuration templates](#) on page 743.

### 7.2.2 Specifying paths to your own format configuration files

To specify the names and locations of format configuration files:

```
[Templates]
; Formats = path to file containing properties for text formats.
Formats = path/to/mytextformats.ini
; Tables = path to file containing properties for table formats.
Tables = path/to/mytableformats.ini
; Pages = path to file containing properties for RTF page layouts.
Pages = path/to/mypagelayouts.ini
; SubFormats = path to .ini file containing properties for subformats.
SubFormats = path/to/mysubformats.ini
```

Your project configuration file should already reference a **DITA2Go**-supplied chain of general configuration templates; see §39.1.3 [Understanding how templates are chained together](#) on page 728. That chain already references the appropriate format template chain. For example, in `d2rtf_config.ini`:

```
[Templates]
Formats=%OMSYSHOME%\d2g\local\formats\local_d2rtf_formats.ini
```

The supplied text-format template in turn references a table-format template, and a format-component template. For RTF output, the text-format template also references a

page-layout template; see §39.1.5 [Understanding how format templates are organized](#) on page 730.

If you want to use *any* of the supplied format definitions, and you have your own text format template (which you reference directly from your project configuration file), that template must reference the supplied chain of format templates. This is because it is an either/or situation: **DITA2Go** builds one format chain or the other, but not both; see §7.2.3 [Understanding how DITA2Go builds format chains](#) on page 111.

In other words: the way you reference your own format templates from your project configuration file is with a `[Templates]Formats` setting. Once you make that setting, any `[Templates]Formats` setting in the general-configuration template chain is null and void. You overrode it. Therefore you must put a reference to the supplied text-format template into your own text-format configuration file, so you still get the supplied settings (except where settings in your own text-format file override them, of course).

If you do *not* reference format template files of your own, you still have access to the supplied templates, as long as your project configuration file references the chain of general configuration templates.

### 7.2.3 Understanding how DITA2Go builds format chains

When **DITA2Go** opens your project configuration file, **DITA2Go** starts with any `[Templates]Configs=` setting, and builds the entire general-configuration template chain before doing anything else. All other template references follow the rules of inheritance for the general configuration chain; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

Next **DITA2Go** looks up the template referenced by `[Templates]Formats`: whichever template is so referenced either in your project configuration file, or failing that, in the configuration template next closest in the general-configuration chain. From this reference **DITA2Go** builds a chain of text-format templates. Not table formats, not page layouts, just text formats.

Next **DITA2Go** looks for references to table-format, format-component, and page-layout templates. **DITA2Go** looks first in the general-configuration template chain, then in the text-format template chain. If a table-format, format-component, or page-layout template is referenced in your project configuration file, or in any general-configuration template of your own, that reference takes precedence over any reference in a text-format template, *even if the text-format template is referenced in your own project configuration file*.

Finally **DITA2Go** builds the chains for table-format, format-component, and page-layout templates. Each such chain consists of only one kind of template; for example, if you try to reference page layouts from a format-component template, **DITA2Go** never sees the reference.

Suppose you use **DITA2Go** to produce Word RTF. In the supplied set of templates, `d2rtf_config.ini` references `local_d2rtf_formats.ini`, but *not* `local_d2rtf_pages.ini`. Then `local_d2rtf_formats.ini` references `d2rtf_formats.ini`, and the latter references `local_d2rtf_pages.ini`.

This means that if you provide your own text-format configuration file, but you want to use the supplied page-format template, you have the following options:

- If you have some use for the supplied text-format template, just link to its editable counterpart from your own new text-format file; the supplied page formats will come along.

- Reference the editable page-format configuration file from your text-format file or from your project configuration file; if you reference it from both, **DITA2Go** uses the reference in your project configuration file.
- Provide your own page-format configuration file and link from it to the supplied editable page-format configuration file.

But if you neither reference the supplied page-format file yourself, nor reference the supplied text-format file that references the page formats, your project will not have access to the supplied page layouts.

## 7.2.4 Understanding how DITA2Go processes format chains

When **DITA2Go** reads the properties of a format, **DITA2Go** does not just read the chain for the configuration section where the format is defined; **DITA2Go** reads the chain for each individual property of that format, across all format files in the chain. This ensures that settings that change defaults for other settings are always processed in the right order, regardless of how you listed them in the format section, thus eliminating one possible source of unexpected results.

When **DITA2Go** reads a format, you can think of it as combining all the sections with that format name, at every template level, with the settings for the lowest level first, then the settings for the next level up, and so on.

If a based property (see §7.4.5 [Basing format properties on other formats](#) on page 116) is anywhere in the combined list, **DITA2Go** takes it into consideration; if the same based property occurs more than once, **DITA2Go** sees just the first instance. Next, **DITA2Go** follows the same process for the format on which the current format is based. Next, if the format is a text format, **DITA2Go** looks for an inline property (see §7.6.4 [Understanding based vs. inline properties for paragraph formats](#) on page 122), and carries out the same recursive process again.

For any given format property setting, you can think of the “search order” like this:

1. The lowest-level instance of the format.
2. The higher-level instances, bottom to top.
3. The inline instances, bottom to top.
4. The based instances, bottom to top.

This can result in quite a few places to look. But it also means that you have a single point of change for common settings, such as font.

If you are in doubt, you can always add the specific property to the lowest-level instance of the format. That overrides any others. But it also means that if you change the value of that property higher in the chain, the change will not propagate to the lower instances.

## 7.3 Creating aliases to existing format names

To use output format names other than the **DITA2Go** default names could require specifying many settings that otherwise you would not have to include in your project configuration file. You can avoid this chore by mapping your preferred names to the **DITA2Go** names.

*In this section:*

§7.3.1 [Understanding reasons for aliasing format names](#) on page 113

§7.3.2 [Mapping legacy names to defined formats](#) on page 113

See also:

[§6.3.4 Mapping wrapper-element outputclass attributes to formats](#) on page 91

[§8.1.3 Assigning additional names to format components](#) on page 142

### 7.3.1 Understanding reasons for aliasing format names

Some of the reasons you might want **DITA2Go** to use output format names other than the default names, or in addition to names of formats you have already defined:

[Legacy format names in outputclass attributes](#)

[Existing CSS classes](#)

[Existing Word template styles](#)

[Non-English input or output.](#)

*Legacy format  
names in  
outputclass  
attributes*

If your DITA document was created from material authored in a non-DITA environment such as FrameMaker, elements might include `outputclass` attribute values that correspond to the format names from the source environment. Rather than substitute those names in many settings, or suppress the use of those names for output formats, you can map the names to **DITA2Go** default format names.

*Existing CSS  
classes*

Suppose you wish to produce HTML output, and you already have a CSS file with classes defined for class names that are different from the **DITA2Go** default output format names. Rather than substitute one of your CSS class names in each of the many settings that assign a format name to an element path, you can prepare a list of aliases that match your class names to those format names.

*Existing Word  
template styles*

If you wish to produce RTF output and you already have a Word template with styles named differently from the **DITA2Go** default format names, you can map those style names to the **DITA2Go** format names.

*Non-English input  
or output*

Another reason to use format names other than the **DITA2Go** default names might be localization; see [§8.9 Localizing output headings, labels, and names](#) on page 157.

### 7.3.2 Mapping legacy names to defined formats

Create a format definition for each of the output format names your project might need. Then you can manage their output appearance in a format template that is integral to your **DITA2Go** project, and that can be shared across many such projects.

To map your names to **DITA2Go** default format names:

```
[FormatAliases]
; format name = another format name; for example:
CellHeading = CellHead
```

You can use this section to specify properties for the format named on the left as though `[FirstFormat]based=SecondFormat` (see [§7.4.5 Basing format properties on other formats](#) on page 116). Make sure that `SecondFormat` is defined somewhere in the format template chain.

When **DITA2Go** looks up format definitions, **DITA2Go** replaces each format name listed on the left with the name assigned to it on the right. This process is not recursive; if you assign format names `a=b` and `b=c`, when **DITA2Go** needs a definition for `a`, instead **DITA2Go** looks for a definition of `b`, not `c`.



## 7.4 Understanding how to define output formats

You define new formats, and override the definitions of default formats, in local or document-specific format configuration files; see §7.2.1 [Understanding where to define output formats](#) on page 110.

*In this section:*

- §7.4.1 [Naming output formats](#) on page 114
- §7.4.2 [Assigning values to format properties](#) on page 115
- §7.4.3 [Documenting output formats](#) on page 115
- §7.4.4 [Understanding the basis of format properties](#) on page 115
- §7.4.5 [Basing format properties on other formats](#) on page 116
- §7.4.6 [Modifying DITA2Go default output formats](#) on page 116
- §7.4.7 [Applying CSS and RTF code to output formats](#) on page 118
- §7.4.8 [Assigning content-adding properties to formats](#) on page 118

### 7.4.1 Naming output formats

A format name must end in a suffix that indicates the type of format (except for the names of paragraph formats, which do not require a suffix). How a format configuration file is referenced in [Templates] settings determines which types of formats are valid in the referenced file, and thus the valid suffixes for the names of formats defined in that file; see §7.2 [Working with format configuration files](#) on page 110. [Table 7-1](#) shows valid suffixes for format names, depending on the [Templates] reference.

**Table 7-1 Valid suffixes for names of formats and format components**

In files referenced via [Templates] setting:	Type of format or component	Format name suffix	Ref.
Formats	Paragraph (block)	Para ( <i>optional</i> )	<a href="#">7.6.3</a>
	Character (inline)	Char	<a href="#">7.6.2</a>
Tables	Table	Table	<a href="#">7.7.2</a>
	Table row	Row	<a href="#">7.7.3</a>
	Table cell	Cell	<a href="#">7.7.4</a>
Pages ( <i>RTF only</i> )	Section	Section	<a href="#">7.8.2</a>
	Page header	Header	<a href="#">7.8.3</a>
	Page footer	Footer	<a href="#">7.8.3</a>
Subformats	Border	Border	<a href="#">8.2</a>
	Shading	Shade	<a href="#">8.3</a>
	Index	Index	<a href="#">14.8.3</a>
	Trademark	Mark	<a href="#">8.8</a>
	Cross reference	Xref	<a href="#">8.7</a>
	Run-in heading	Head	<a href="#">8.6</a>
	Numbering	Num	<a href="#">8.5.3</a>
	Number stream	Stream	<a href="#">8.5.2</a>

**DITA2Go** validates formats partly by suffix. A wrong type could result in very odd output, plus a warning in the log file.



## 7.4.2 Assigning values to format properties

To define an output format, create a section named for the format in the appropriate format configuration file. The settings in this section are all of the form `property = value`:

```
[FormatName]
property1 = value1
property2 = value2a value2b value2c ...
. . .
```

The collection of properties you can specify depends on which kind of format you are defining: paragraph, character, component, table, or page. For example, for a paragraph format named `Heading1` you might specify the following properties:

```
[Heading1]
based = Body
section = Section
keep = next
break = page
font size = 20pt
font weight = bold
margin top = 18pt
margin bottom = 6pt
```

You can use macros in output-format settings. However, you cannot *define* macros in a format configuration file; you must define them in macro libraries. See §37.2.4 [Including macro definitions in your own macro library](#) on page 685.

## 7.4.3 Documenting output formats

To make it easy to identify and locate format definitions, you can assign values to two format meta properties:

```
[FormatName]
category = CategoryName
help = Any text to remind you of the nature and purpose of the format
```

The value for `category` must be the name of one of the format subcategories defined in the Configuration Manager. See §3.2.5 [Choosing a configuration category or file type](#) on page 54. However, if you have named your format according to the suffix requirements described in §7.4.1 [Naming output formats](#) on page 114, you do not need to include a `category` setting in the definition; the Configuration Manager will deduce the category from the suffix.

The value for `help` can be any text that will help you remember what the format is for. The Configuration Manager displays `help` text for every format.

## 7.4.4 Understanding the basis of format properties

The properties you can specify for output formats are based on CSS rules, even if you are generating RTF output. If you are not familiar with CSS (Cascading Style Sheets), see §31.2 [Understanding how to use CSS](#) on page 591.

**DITA2Go** augments CSS rules with the following features:

- [Number streams](#) (allowing more complexity than CSS permits)
- [Based](#) formats, for incorporating properties of another format
- Output-type-specific properties such as [CSS](#) and [RTF](#)
- Content-adding properties such as [Start](#), [End](#), [Before](#), [After](#), [Replace](#).

<i>Number streams</i>	Format numbering is handled in the <code>SubFormat</code> templates. The idea is to provide building blocks, so you do not have to repeat the numbering properties in every block format that uses them.
<i>Based</i>	Format property <code>based</code> allows you to base a format on the properties already defined for another format. See §7.4.5 <a href="#">Basing format properties on other formats</a> on page 116.
<i>CSS</i>	Format property <code>css</code> allows you to assign any CSS to a format, for HTML output.
<i>RTF</i>	Format property <code>rtf</code> allows you to assign any RTF control words to a format, for RTF output. See §7.4.7 <a href="#">Applying CSS and RTF code to output formats</a> on page 118.
<i>Start, End, Before, After, Replace</i>	Format properties <code>start</code> , <code>end</code> , <code>before</code> , and <code>after</code> add text or code around every instance of a format, and <code>replace</code> substitutes text or code for every instance. See §7.4.8 <a href="#">Assigning content-adding properties to formats</a> on page 118.

## 7.4.5 Basing format properties on other formats

Format property `based` confers all the properties of the referenced format on the current format.

Suppose you have based a given format on some other format:

```
[FormatName]
based = OtherFormat
```

This setting says: if a property is not specified in `[FormatName]`, **DITA2Go** should get it from the properties of `OtherFormat`. However, before doing so, **DITA2Go** first looks for higher-level (farther along the template chain) instances of `FormatName`; and if found, if the property in question is defined there, uses that value for the property. The implication is that if you are going to base a format on another one, there should be no higher-level instances of the basing format.

You can chain based formats. When `based=parent` **DITA2Go** goes through the `based` chain to the end, or to the first instance of a format already seen (to break circularity), just as with configuration templates.

**Note:** If you override a property of a format that other formats are based on, and it is a property that those other formats do *not* override, the change propagates to all those basing formats.

Because format properties are assigned in fixed-key configuration sections (see §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769), they inherit from higher-level (farther away in the chain of templates) sections of the same name. For example, if you override a property that sets `margin left = 6pt`, and you want *no* left margin, you have to specify `margin left = 0` explicitly, just as you would have to do to override a property of the format on which you based a new format.

However, if the overridden format at the higher level is itself based on yet another format, and the new format is also based, the new format does *not* inherit the properties of the format the higher-level format is based on, just the properties set in the overridden format.

## 7.4.6 Modifying DITA2Go default output formats

To change the definition of a format supplied in the **DITA2Go** format templates, include a section for that format in the corresponding `local_` format configuration file or your own format configuration file, and override the default values of any properties you wish to change. Your version of the format will inherit all the properties that you do *not* redefine. Because many of the supplied formats also inherit from other formats, you will have to

follow the chain for each inherited property of interest until you find its actual value, then decide how to override that value.

For example, suppose you want to change the vertical alignment of text in header cells in tables that you have assigned format `SeaTable`, for RTF output. You would look in format configuration template `d2rtf_tables.ini` for the default definition:

```
[SeaTable]
based = NormalTable
border top = NoBorder
border bottom = NoBorder
header rows = SeaHeaderRow
body rows = SeaBodyRow
```

This definition tells you that header rows are in format `SeaHeaderRow`; so you look for that definition:

```
[SeaHeaderRow]
based = HeaderRow
body cells = SeaColHeadCell
border bottom = NoBorder
shading = SeaHeadShade
```

Now you see that the cells of the header row are defined in format `SeaColHeadCell`, so you look for that definition:

```
[SeaColHeadCell]
based = ColHeadCell
border bottom = NoBorder
shading = SeaHeadShade
```

Now you find that `SeaColHeadCell` does not have any alignment properties defined, but instead is based on `ColHeadCell`, so you look there:

```
[ColHeadCell]
based = BodyCell
margin top = 1pt
margin bottom = 3pt
border bottom = LightBorder
vertical align = Bottom
```

Finally you have found the properties that govern vertical alignment of text in header cells! You might not want to change the definition of `ColHeadCell`, because that definition might be inherited by table formats other than `SeaTable`. But you can be fairly sure that `SeaColHeadCell` is specific to `SeaTable`, just based on its name. To be absolutely sure, you would check for any inheritance of `SeaColHeadCell` by any other table or row format in the current template or in any table format configuration file “downstream” of that template.

Now you can redefine `SeaColHeadCell` as you wish, overriding only those properties you want changed: namely, `vertical align` (and possibly `margin top` and `margin bottom`), inherited from `ColHeadCell`:

```
[SeaColHeadCell]
vertical align = top
```

Any tables in your document that are assigned format `SeaTable` will now have header cells with text alignment toward the top of the cell instead of the bottom, within the confines of the relevant margin settings.

In this particular example, all the inherited properties are defined in the same format configuration template. However, to modify some formats you might have to look further “upstream” in other templates to find the properties you want to change. You would start

looking in the format configuration file referenced by your project configuration file, and work back from there.

### 7.4.7 Applying CSS and RTF code to output formats

You can apply CSS code or RTF control words to all instances of a text or table format:.

[Apply CSS code with property `css`](#)

[Apply RTF code with property `rtf`](#).

You can even assign both properties `css` and `rtf` to the same format.

*Apply CSS code  
with property `css`*

Values for property `css` take the usual CSS form, and must follow the usual CSS rules. Each `css` setting must either fit on a single line or reference a macro. Within a `css` value, observe the following:

- To include a line break, use `\n`. For example:  

```
[FormatName]
css = padding: 2 6 4 0;\n width: 80%;
```
- To retain a backslash in CSS, double it.
- To add a leading or trailing space, use `\~`. Normal leading and trailing spaces are removed.

*Apply RTF code  
with property `rtf`*

Values for property `rtf` use the usual RTF control words. Each `rtf` setting must either fit on a single line, or reference a macro. Within an `rtf` value, observe the following:

- To retain a backslash in an `rtf` setting, double the backslash. For example:  

```
[FormatName]
rtf = \\lang1033 \\widctlpar\\noproof
```
- To add a leading or trailing space, use `\~`. Normal leading and trailing spaces are removed.

### 7.4.8 Assigning content-adding properties to formats

You can use properties `start`, `end`, `before`, and `after` to add text or code around every instance of a format; and property `replace` to substitute text or code for each instance. All five apply to text formats; `before`, `after`, and `replace` apply to table formats also.

These properties are not based on CSS, because many current browsers (notably Internet Explorer) do not support the CSS settings required. Therefore, **DITA2Go** adds the code for these properties directly to HTML output.

The content-adding properties provide alternatives to the corresponding `[StyleCode*]` sections in your project configuration file. For example:

With a `[StyleCode*]` section:

```
[WordStyles]
isbn = CodeStart

[StyleCodeStart]
isbn = ISBN:
```

With format properties:

```
[isbn]
start = ISBN:
```

You can include macros in the content-adding settings; see §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711. Each setting must either fit all on one line, or reference a macro. Any regular configuration settings in `[StyleCode*]` sections in

your project configuration file, or in a general configuration template, override content-adding properties.

[Table 37-10](#) on page 712 shows exactly where each added content is placed in output.

Some examples for HTML output:

```
[FormatName]
before = <h3 class="caution">Caution!</h3>
start = <i>For example: </i>
end = <$MyEndingMacro>
after = <hr style="width:50%;"/>
```

Some examples for RTF output:

```
[FormatName]
before = \\pard \\s<$_stylenum("CautionFormat")> Caution!\\par
start = {\\i For example:\\~ }
end = <$MyEndingMacro>
after = \\pard\\li2880\\ri2880\\brdrb\\brdrw20\\brdrsp120\\par
```

For RTF content, you can also use the following predefined macros:

<code>&lt;\$_style(stylename)&gt;</code>	RTF start code for the paragraph or character format; mainly for setting up title pages with a macro used after the title itself
<code>&lt;\$_colornum(colorref)&gt;</code>	Just the color number, used after <code>\\cf</code> or <code>\\cb</code> (foreground or background)
<code>&lt;\$_fontnum(fontname)&gt;</code>	Just the font number, used after <code>\\f</code>
<code>&lt;\$_stylenum(stylename)&gt;</code>	Just the style number; use it after <code>\\s</code> in your own RTF code
<code>&lt;\$_styleref(stylename)&gt;</code>	STYLEREF field with the named style, useful in headers and footers (only)
<code>&lt;\$_pageref&gt;</code>	PAGEREF field referencing last bookmark used, for after TOC formats

For example:

```
[TOC]
based = Heading2
font size = 13pt
line height = 16pt
margin top = 8pt
section = FrontSection
tabs = dot R3in
end = \\tab \\~<$_PageRef>
```

## 7.5 Understanding text output formats

Text output formats include block (paragraph) and inline (character) formats. **DITA2Go** provides two sets of properties for output paragraph and character formats:

- Inline properties that apply to both character and paragraph formats
- Block properties that apply only to paragraph formats.

*In this section:*

- §7.5.1 [Understanding where to define text output formats](#) on page 120
- §7.5.2 [Providing padding around the text body area for HTML](#) on page 120
- §7.5.3 [Establishing base values and units of measurement](#) on page 120
- §7.5.4 [Including formats for features not present in body content](#) on page 120

## 7.5.1 Understanding where to define text output formats

Define text formats in a format configuration file, referenced from your project configuration file (or from another template) as follows:

```
[Templates]
; Formats = path to file containing properties for text formats.
Formats = path/to/mytextformats.ini
```

See §7.2 [Working with format configuration files](#) on page 110.

## 7.5.2 Providing padding around the text body area for HTML

To specify padding around text in HTML output:

```
[BodyElement]
css= padding: 6pt;
```

This implied format for the HTML `<body>` element (not the Body format) sets the padding inside the display window.

## 7.5.3 Establishing base values and units of measurement

The numerical properties of a given text format can be expressed relative to a base value. You can establish base values for calculating relative sizes of properties with units of em, en, or percent. These base values must be specified with one of the following fixed units:

```
px    (pixels)
pt    (points)
pc    (picas)
cm    (centimeters)
mm    (millimeters)
in    (inches)
```

To specify a base size for text formats:

```
[BaseValues]
; BodyBaseHeight = size to use as basis for calculating relative
; values for font size, line height, and all values using units
; of em or ex. If not set, 10pt is used.
BodyBaseHeight=10pt
```

To specify a base size of the containing element (usually body) for margins, padding, and indentation:

```
[BaseValues]
; BodyBaseWidth = width to use as basis for calculating percent
; values for margins, padding, and text-indent. If not set, 6in
; is used.
BodyBaseWidth=6in
```

Percent values are tricky; you might want to avoid their use unless you are very familiar with advanced CSS.

## 7.5.4 Including formats for features not present in body content

You can force inclusion of formats that are not actually used in the body content of the source, such as those employed only for headers and footers. For example:

```
[Required]
; format name = format type, block or inline
```

```

Author = block
Header = block
Footer = block

```

For HTML output, this forces inclusion of formats that are not defined in the source file, such as those used in macros. Because content in these formats is added at the end of processing, formats listed here cannot contribute any before, after, start, or end content to HTML output.

For RTF output, if you do not specify a format type the correct properties are still used, but they appear as overrides to format `Normal`.

## 7.6 Configuring text output formats

The **DITA2Go**-supplied format templates include basic default properties for character and paragraph formats. If your project configuration file references the chain of format templates, these properties will be in effect for any text formats you define. You can change the defaults by including your own default values in your own format configuration file, referenced via `[Templates]Formats`; see [§7.2 Working with format configuration files](#) on page 110.

*In this section:*

- [§7.6.1 Naming and defining text formats](#) on page 121
- [§7.6.2 Specifying default properties for character formats](#) on page 121
- [§7.6.3 Specifying default properties for paragraph formats](#) on page 122
- [§7.6.4 Understanding based vs. inline properties for paragraph formats](#) on page 122
- [§7.6.5 Specifying inline properties for paragraph and character formats](#) on page 123
- [§7.6.6 Specifying block properties for paragraph formats](#) on page 124
- [§7.6.7 Configuring list formats](#) on page 125
- [§7.6.8 Assigning border properties to paragraph formats](#) on page 128
- [§7.6.9 Configuring character formats for HTML links](#) on page 128
- [§7.6.10 Specifying tab positions for RTF paragraph styles](#) on page 129

### 7.6.1 Naming and defining text formats

Define text formats in a text-format configuration file referenced from your project configuration file (or from another text-format template) as follows:

```

[Templates]
; Formats = path to file specifying properties of text formats.
Formats = path/to/mytextformats.ini

```

See [§7.2 Working with format configuration files](#) on page 110.

When you define a new character format, give it a name that ends in `Char`. The `Char` suffix is required for character format names. Any format defined in a configuration file accessed via `[Templates]Formats` is considered to be a paragraph format unless the name ends in `Char`.

The conventional suffix for paragraph format names is `Para`; however, paragraph format names do not require a suffix.

### 7.6.2 Specifying default properties for character formats

You can establish default properties for character formats, and then use the `based` property to apply those properties to other character formats.



To specify default properties for character output formats (for example):

```
[Char]
help = Default character format for an inline element
category = CharacterFormats
display = inline
; Use Verdana; if not available, use another sans serif font:
font name = Verdana, sans serif
; Make the characters 10 points in size:
font size = 10pt
```

Character format `Char` is the default format for any inline element to which a format name has not been assigned. The property values listed here for `Char` are the default values. Unless you want to specify a different value for a particular property, you do not have to include these property settings for a character format you define.

### 7.6.3 Specifying default properties for paragraph formats

The **DITA2Go**-supplied format templates include basic default properties for character and paragraph formats. If your project configuration file references the chain of format templates, these properties will be in effect for any text formats you define. You can change the defaults by including your own settings in the corresponding `local_` format configuration file or in your own format configuration file, referenced via `[Templates]Formats`; see §7.2 [Working with format configuration files](#) on page 110.

To specify default properties for paragraph formats (for example):

```
[Para]
help = Default paragraph format for a block element
category = ParagraphFormats Configuration Manager text-format category
display = block This is a block format.
inline = Char Use the character properties of format Char.
line height = 11pt Make leading 11 points.
```

Paragraph format `Para` is the default format for any block element to which you have not assigned a format name.

**DITA2Go** applies each of the values specified for `Para`, and also for default character format `Char` (because `Para` references `Char` via property `inline`), to any paragraph format for which you do not specify a different value for the property in question.

The property values listed in `[Char]` and `[Para]` for inline and block properties, respectively, are the default values. Unless you want to specify a different value for a particular property, you do not have to include any of these property settings for a text format.

### 7.6.4 Understanding based vs. inline properties for paragraph formats

Properties `inline` and `based` are almost the same; however, `inline` is a property only of paragraph formats, and its value must be the name of a character format. **DITA2Go** processes `based` first, then `inline`. Therefore you can use the `based` property to base one paragraph format on another, then change just the character properties of the first paragraph by applying the `inline` property. For example:

```
[SomePara]
display = block
based = OtherPara
inline = SomeChar
```

With these settings, `SomePara` would have all the characteristics of `OtherPara`, except for the character properties of character format `SomeChar`.



## 7.6.5 Specifying inline properties for paragraph and character formats

The properties listed in [Table 7-2](#) define the appearance of text formats in both HTML and RTF output.

**Table 7-2 Inline properties for text formats**

Property name	Valid property values
based	Name of another character format
font family	One or more font names per CSS rules. For outputs which allow only single fonts, such as RTF, the first font name is used. For RTF output, <i>do not</i> enclose the font name in quotes
font name	Synonym for font family
font size	Size; see §7.5.3 <a href="#">Establishing base values and units of measurement</a> on page 120
font style	normal, italic, or oblique
font variant	normal or small-caps
font weight	normal or bold
font kern	Percent or points or other units: positive number to expand text, negative number to compress text
text decoration	none, or a combination of underline, overline, and strike-through
text transform	none, uppercase, lowercase, or capitalize
color	CSS color designation
background color	CSS color designation
selector	(HTML output only) Format name preceded by a dot, for CSS use; overrides the default .class

For example:

```
[Bold]
based = Char
font weight = bold

[Italic]
based = Char
font style = italic
```

Bold and Italic both incorporate the properties of character format Char; see §7.6.2 [Specifying default properties for character formats](#) on page 121.

To specify text color for a heading (for example):

```
[MySubHeading]
based = Heading2
color = rgb(42,96,148)
```

To show a horizontal line above the heading, you could use either a block-property fixed-length line in any color, or an inline-property overline in the same color as the text. For the inline property you would add:

```
text decoration = overline
```

For the block property, you would instead specify a top border for the paragraph:

```
border top = MyBorder
```

And then define the border format:

```
[MyBorder]
color = rgb(42,96,148)
type = single
```

```
thick = 1pt
space = 6pt
```

See §8.2 [Defining border format components](#) on page 144.

To compress text in a paragraph format using a percent value for font kern:

```
[CellCodePara]
based = CodePara
font size = 9pt
font kern = -5%
```

## 7.6.6 Specifying block properties for paragraph formats

Paragraph formats can reference character formats for their inline properties, using property `inline`, and can also specify inline properties explicitly. For paragraph formats, use the properties listed in [Table 7-3](#), in addition to those listed in [Table 7-2](#).

**Table 7-3 Block properties for output paragraph formats**

Type of property	Property name	Valid property values
Inherited	<code>inline</code>	Name of a character format
	<code>based</code>	Name of another paragraph format
Code	<code>rtf</code>	( <i>RTF only</i> ) Any valid RTF control sequence; see §7.4.7 <a href="#">Applying CSS and RTF code to output formats</a> on page 118
	<code>css</code>	( <i>HTML only</i> ) Any applicable CSS code; see §7.4.7 <a href="#">Applying CSS and RTF code to output formats</a> on page 118
Lists: sequential numbering or bulleting	<code>list level</code>	An integer to indicate the nesting or indentation level; 0 means no indentation
	<code>list style</code>	( <i>HTML only</i> ) See §7.6.7.1 <a href="#">Assigning properties to list formats for HTML list styles</a> on page 125
	<code>number</code>	Name of a number or symbol format component to apply; see §8.5.3 <a href="#">Defining number format components</a> on page 148; or <code>None</code> .
	<code>numref</code>	( <i>Footnote formats only</i> ) Name of a number format component to use for footnote references; see §8.5.3 <a href="#">Defining number format components</a> on page 148
Run-in heading	<code>runin</code>	Name of a format component that defines text to precede the paragraph; see §8.6 <a href="#">Configuring run-in headings for text formats</a> on page 153; or <code>None</code> .
Cross reference	<code>xref</code>	Name of a cross-reference format to use for references to text in this paragraph format; see §8.7 <a href="#">Defining cross-reference output formats</a> on page 155
Positioning relative to other blocks ( <i>RTF output only</i> )	<code>section</code>	Name of section format in page template file to be used whenever this paragraph style appears; see §7.8.2 <a href="#">Configuring output section formats</a> on page 135.
	<code>section start</code>	No (default, start section only if not already in it), or Yes (always restart section)
	<code>section break</code>	none, column, page, left, right
	<code>break</code>	none, column, page, left (page), or right (page)
	<code>keep</code>	none, or together and/or next
Alignment	<code>text align</code>	left, center, right, or justify
	<code>vertical align</code>	top, middle, or bottom

**Table 7-3 Block properties for output paragraph formats (continued)**

Type of property	Property name	Valid property values
Line and margin dimensions	line height	Size can be absolute or relative; see §7.5.3 <a href="#">Establishing base values and units of measurement</a> on page 120.
	margin left	
	margin right	
	margin top	
	margin bottom	
	text indent	
	padding left	
	padding right	
	padding top	
	padding bottom	
Tab positioning ( <i>RTF output only</i> )	tabs	Tab definitions; see §7.6.10 <a href="#">Specifying tab positions for RTF paragraph styles</a> on page 129.
Border around text	border <i>position</i>	Names of border formats; see §7.6.8 <a href="#">Assigning border properties to paragraph formats</a> on page 128
Background color	shading	Name of a shade format; see §8.3 <a href="#">Defining shading format components</a> on page 145
CSS ( <i>HTML output only</i> )	selector	Format name preceded by a dot, for CSS use; overrides the default <code>.class</code>

## 7.6.7 Configuring list formats

**DITA2Go** provides two ways to configure block formats for paragraphs that form lists, such as a series of numbered steps or a series of bulleted items. For HTML-based output, you can use CSS/HTML list styles (the default); or, for any type of output, you can roll your own using **DITA2Go** list properties. Or you can configure both ways, and choose which to use later for a given output.

*In this section:*

§7.6.7.1 [Assigning properties to list formats for HTML list styles](#) on page 125

§7.6.7.2 [Assigning properties to list formats for DITA2Go list styles](#) on page 127

§7.6.7.3 [Having the best of both worlds](#) on page 127

### 7.6.7.1 Assigning properties to list formats for HTML list styles

For HTML-based output, **DITA2Go** uses HTML list styles by default; see §30.11.2 [Converting list formats to HTML list styles](#) on page 585. If you are using CSS, in a formats configuration file you can assign any or all of three list-style properties to a paragraph format that begins a list:

```
[FormatName]
list style = type image position
```

where:

<i>type</i>	can be one of the following:
	disc (a filled-in circle; standard bullet)
	circle
	square
	decimal
	decimal-leading-zero
	upper-alpha (upper-latin)
	lower-alpha (lower-latin)
	upper-roman
	lower-roman
	lower-greek
	armenian
	georgian
	none (no number, letter, or symbol)
<i>image</i>	is a URL, such as <code>graphic.gif</code>
<i>position</i>	can be one of the following:
	outside (default)
	inside (within the same indent as the text)

For example:

```
[Numbered1First]
based = Numbered
list style = decimal

[Bulleted]
list style = circle mybullet.gif
```

If you specify values for both *type* and *image*, but the browser cannot locate the image, the *type* value is displayed instead. It is a good idea always to include a value for the *type* property, even if you really want *image*.

If you omit a `list style` property, you get the browser defaults for the list type (`ol` or `ul`). If you set `list style=none`, any list keywords you assigned to the format in `[HTMLParaStyles]` are disabled; see §30.11.2.1 [Specifying HTML list styles \(deprecated\)](#) on page 585. Use `list style=none` for continuation paragraphs that should be included in the same `<li>` element.

#### Padding and margins

To achieve some degree of cross-browser indentation consistency, you can adjust padding and margins for list elements, by redefining the following predefined formats:

```
[OrderedListElement]
margin left = 18pt
padding left = 12pt

[UnorderedListElement]
margin left = 18pt
padding left = 12pt
```

These formats are for the `<body>` element, not the `Body` format. They affect all the elements of their CSS selectors regardless of class, thus providing a kind of baseline rule. This rule overrides the built-in CSS rule in a browser.

#### Indentation

To specify indentation for list formats:

```
[FormatName]
list level = N
```

The value *N* is an integer from 0 to however deep you nest lists. When `list level=0`, the number or symbol that marks the list item is not indented except for a small amount to keep it within the page margin, and the text of the list item is indented the amount

specified for `margin left` in `[OrderedListElement]` or in `[UnorderedListElement]`. For each successive value of `list level`, the list item is indented a similar (cumulative) amount relative to the page margin.

For example:

```
[Numbered2]
based = Numbered1
list style = lower-alpha
list level = 2
```

Indentation using CSS is not just a matter of specifying distance from the left margin. It is more like Petroushka dolls: one element fits inside another. If you do not use lists, you can overlook that, because all the block elements are in the same “box”: the `<body>` element. With lists, you have an ordered-list `<ol>` or unordered-list `<ul>` box inside the `<body>` box, with list-item `<li>` boxes inside that; all the list content is in the `<li>` boxes. This means that for list content, all margins are relative to `Numbered` or `Bulleted` in the current list, so most of them will have `margin left = 0` to align with the numbered or bulleted *text* (not the number or bullet).

We find that *Bulleted* formats need `margin left = 0` at all levels; *Numbered* formats need `margin left = 2pt` at all levels. The 2pt is needed to make the number align with the item above it; bullets are closer to the text, and do not need the extra indent.

For a detailed explanation of CSS list properties, see Chapter 12 of the O'Reilly book “CSS: The Definitive Guide”, 3rd Edition, by Eric Meyer. For cautionary statements about the pitfalls of using list styles for HTML output, see §30.11 [Converting list formats to HTML \(deprecated\)](#) on page 584.

### 7.6.7.2 Assigning properties to list formats for DITA2Go list styles

The main benefit of **DITA2Go** list styles is the ability to use any characters as bullets, rather than the limited assortment supported by CSS; see §7.6.7.1 [Assigning properties to list formats for HTML list styles](#) on page 125.

**DITA2Go** list styles are the default for RTF output. For HTML-based output, specify the appropriate settings described in §30.11.2 [Converting list formats to HTML list styles](#) on page 585.

To specify margins and indentation for **DITA2Go** list styles:

```
[ListOptions]
LevelMargin = 12pt
NumberOutdent = 8pt
BulletOutdent = 6pt
```

To specify numbering for ordered lists, see §8.5 [Configuring output numbering properties](#) on page 146. To specify bullets for unordered lists, define number formats with `stream=none` and `form=` a Unicode value; see §8.5.5.4 [Specifying symbols for bulleted lists](#) on page 152.

### 7.6.7.3 Having the best of both worlds

For HTML-based output, if you are not using CSS, the `list style` property does not apply. Instead, **DITA2Go** uses the format component assigned to property number; see [Table 7-3](#) on page 124. The `list style` property is for CSS, and determines what an ordered or unordered list looks like. List styles do not produce anything in RTF output.

The number property, in contrast, produces an autonumber. Autonumbers work in Word and also in HTML, outside of lists; within HTML list styles, **DITA2Go** turns off autonumbers by default.

It is a good idea to set both number and list style, so that your formats are covered whichever way you go. You will not get duplicate numbering if you specify values for both list style and counter types. For example:

```
[Numbered2]
based = Numbered
list style = lower-alpha
number = List2Num

[List2Num]
stream = ListStream
counter = 2
form = <tab/><counter2/>.<tab/>

[ListStream]
counters = 3
counter types = Num LCAAlpha LCRom
```

See also §8.5.5 [Considering examples of numbering schemes](#) on page 150.

## 7.6.8 Assigning border properties to paragraph formats

To provide a border around output text, assign one or more border properties to the paragraph format of the text, and assign the name of a border format component to each property. [Table 7-4](#) lists the border properties you can assign to a paragraph format.

**Table 7-4 Border properties for paragraph formats**

Border property	Position relative to text
border top	Above the paragraph
border bottom	Below the paragraph
border left	To the left of the paragraph
border right	To the right of the paragraph
border outer	(RTF only) Toward the outer edge (like change bars)
border box	(RTF only) All the way around a paragraph
border between	(RTF only) Where multiple paragraphs have borders; by default, there are no borders between adjacent bordered paragraphs, unless you specify this border format for those paragraphs

Each border property setting specifies the name of a border format component to use for that property for that paragraph format; for example:

```
[StandOutPara]
border box = HeavyBorder
```

Border format components are defined in Subformats configuration files; see §8.2 [Defining border format components](#) on page 144.

## 7.6.9 Configuring character formats for HTML links

In addition to formats you can name and define, **DITA2Go** provides three predefined character formats to use for HTML links. You can redefine these formats:

```
[AElement]
; Turn off link underlining:
text decoration = none
```

```
[AUnvisitedElement]
; Make unvisited links blue:
color = blue

[AVisitedElement]
; Make visted links purple:
color = purple
```

These formats affect all the elements of their CSS selectors regardless of class, thus providing a kind of baseline rule. This rule overrides the built-in CSS rule in a browser.

### 7.6.10 Specifying tab positions for RTF paragraph styles

A value for the `tabs` setting consists of the following:

- an optional leader, one of: `dot`, `hyph`, `ul` (underline), `th` (thick), or `eq` (equals), followed by a space
- a letter for the tab type (if other than left): `R` (right), `C` (center), `D` (decimal), or `B` (bar)
- a numerical value followed by the units for the distance of the tab from the left margin.

For example, you would specify `tabs=dot R6.75in` for a flush-right tab with a dotted leader in a body frame 6.75 inches wide. If you do not include a letter, a left tab is assumed. You can specify as many space-delimited tabs as you want, but all must either fit on a single line, or be included in a referenced macro.

## 7.7 Configuring table output formats

To configure the appearance of tables in output, you can define individual table formats, row formats, and cell formats. Table format properties can reference row formats and cell formats. Row format properties can reference cell formats. This arrangement allows for maximum flexibility and excruciating complexity in the display properties of tables.

*In this section:*

§7.7.1 [Naming and defining table, row, and cell formats](#) on page 129

§7.7.2 [Configuring table format properties](#) on page 130

§7.7.3 [Configuring row format properties](#) on page 131

§7.7.4 [Configuring cell format properties](#) on page 132

*See also:*

§6.9 [Specifying formats and options for tables](#) on page 103

§6.3.2 [Mapping table outputclass attributes to formats](#) on page 90

§15.6 [Converting tables to print RTF](#) on page 232

§17.5 [Converting tables to WinHelp RTF](#) on page 290

§33 [Converting tables to HTML](#) on page 625.

### 7.7.1 Naming and defining table, row, and cell formats

Define output formats for tables in a table-format configuration file referenced from your project configuration file (or from another table or text format template) as follows:

```
[Templates]
; Tables = path to file specifying properties of table formats.
Tables = path/to/mytableformats.ini
```

See §7.2 [Working with format configuration files](#) on page 110.

**Note:** Table specifications in a format configuration file can be overridden by other settings for tables in general configuration files.

When you define a new table format, give it a name that ends in `Table`. The `Table` suffix is required for table format names. Names of table row formats must end in `Row`. Names of table cell formats must end in `Cell`.

Any format defined in a configuration file accessed via `[Templates]Tables` is considered to be a table format unless the name ends in `Row` or `Cell`.

## 7.7.2 Configuring table format properties

As a convention, the name of any table format should end in `Table`. The default table format is named `Table`. Table format `Table` has no default properties; however, you can define a format for `Table`:

```
[Table]
property1 = value1
property2 = value2a value2b value2c ...
. . .
```

For HTML output, table format names are used in the `class` attribute of table elements, and result in CSS that implements their properties. [Table 7-5](#) lists the properties you can define for tables.

**Table 7-5 Table output format properties**

Table property	Valid table property values
<code>category</code>	<code>TableFormats</code>
<code>help</code>	Any text to describe purpose or use of format
<code>table align</code>	<code>left</code> , <code>center</code> , or <code>right</code>
<code>border model</code>	<code>collapse</code> or <code>separate</code>
<code>border spacing</code>	( <code>separate model</code> only) horizontal then vertical number and units
<code>empty cells</code>	( <code>separate model</code> only) <code>show</code> or <code>hide</code> borders and background
<code>table layout</code>	<code>auto</code> or <code>fixed</code>
<code>margin top</code>	Space above the table top border
<code>margin bottom</code>	Space below the table bottom border
<code>margin left</code>	Distance from the page margin to the left edge of the first column
<code>table width</code>	Number and units of the size on which percent column widths are based
<code>column widths</code>	List of column widths from left to right; if the column count exceeds the list length, the last value is reused; if the width is in percent, the table width must be specified
<code>column border left</code> <code>column border right</code>	Border format names for left and right borders of columns; default left and right borders for cells in each column; overridden by cell borders and table borders
<code>cell margin left</code> <code>cell margin right</code> <code>cell margin top</code> <code>cell margin bottom</code>	( <i>RTF only</i> ) Number and units; default values for all cells; can be overridden by cell format properties
<code>cell padding left</code> <code>cell padding right</code> <code>cell padding top</code> <code>cell padding bottom</code>	( <i>HTML only</i> ) Number and units; default values for all cells; can be overridden by cell format properties
<code>header rows</code>	List of names of header row formats, in top-down order of header rows; if the number of header rows exceeds the list length, the last format is reused



**Table 7-5 Table output format properties (continued)**

Table property	Valid table property values
body rows	List of names of body row formats, in top-down order of body rows; if the number of body rows exceeds the list length, the whole list is restarted
footer rows	List of names of footer row formats; if the number of footer rows exceeds the list length, the last format is reused
border left border right border top border bottom	Names of border formats; properties of these formats override row and cell border properties
shading	Shading format, overridden by row and cell shading
css	(HTML only) Adds information to the table format CSS code
rtf	(RTF only) Adds information to the table format (at the start of each row)
before	Adds content before the table
after	Adds content after the table

Some examples:

```
[Table]
table align = left
margin left = 0
cell margin left = 1pt
cell margin right = 1pt
cell margin top = 1pt
cell margin bottom = 1pt
header rows = HeaderRow
body rows = Row

[NormalTable]
based = Table
margin top = 3pt
margin bottom = 6pt
table width = 7in
column widths = 3in 2in 2in
column border left = LightBorder
column border right = LightBorder
header rows = HeaderRow
body rows = BodyRow BodyRow ShadedRow
border top = LightBorder
border bottom = LightBorder
border left = NoBorder
border right = NoBorder
```

### 7.7.3 Configuring row format properties

The names of table row formats must end in `Row`. The default row format is named `Row`. Row format `Row` has no default properties; however, you can define a format for `Row`:

```
[Row]
property1 = value1
property2 = value2a value2b value2c ...
...
```

For HTML output, row format names are used in the `class` attribute of row elements, and result in CSS that implements their properties. [Table 7-6](#) lists the properties you can define for rows.

Columns are handled by cell lists in row formats. You must keep those lists synchronized for all row formats referenced in a table format, or it gets ugly. For example, you must

have the same count of row header cells in all of the row formats referenced by the table format.

**Table 7-6 Format properties of table rows**

Row format property	Valid format property values
category	TableRowFormats
help	Any text to describe purpose or use of format
row type	body, header, or footer; overrides the row type specified in the source document
row height	adapt (to text, default), or minimum (adapts to text, but has the minimum size given)
keep	none (allow page break in row) or together; RTF output only
header cells	List of names of cell formats for row header cells; the number of formats in the list is the number of cells used as row headers
body cells	List of names of cell formats for body cells, forming a column pattern; if the number of body cells exceeds the list length, the list is restarted
border left border right border top border bottom border sep	Names of border formats; table and cell border properties override row border properties; when two row borders conflict, the heavier border prevails; border sep is used between header and body rows, and between footer and body rows, overriding top or bottom borders.
shading	List of shading formats; overrides table shading; overridden by cell shading
css	(HTML only) Adds information to the row-format CSS code
rtf	(RTF only) Adds information to the row-format RTF code

Some examples:

```
[Row]
; These are the default row properties:
row type = body
body cells = Cell
border bottom = LightBorder

[BodyRow]
based = Row
body cells = BodyCell BodyCell ShadedCell
border top = LightBorder
border bottom = LightBorder
border left = LightBorder
border right = LightBorder

[HeaderRow]
based = BodyRow
row type = header
body cells = ColHeadCell
border bottom = HeavyBorder
border sep = DoubleBorder

[ShadedRow]
based = BodyRow
body cells = ShadedCell ShadedCell XtraShadedCell
shading = LightGreyShade
```

## 7.7.4 Configuring cell format properties

The names of cell formats must end in `Cell`. The default cell format is named `Cell`. Cell format `Cell` has no default properties; however, you can define a format for `Cell`:

```
[Cell]
property1 = value1
property2 = value2a value2b value2c ...
. . .
```

For HTML output, cell format names are used in the `class` attribute of cell elements, and result in CSS that implements their properties. [Table 7-7](#) lists the properties you can define for cells.

**Table 7-7 Format properties of table cells**

Cell format property	Valid cell property values
category	TableCellFormats
help	Any text to describe purpose or use of format
vertical align	top, middle, or bottom
margin left	(RTF only) Number and units; values override corresponding cell margin settings in table formats
margin right	
margin top	
margin bottom	
padding left	(HTML only) Number and units; values override corresponding cell padding settings in table formats
padding right	
padding top	
padding bottom	
border left	Names of border formats; override row but not table borders
border right	
border top	
border bottom	
shading	Shading format; overrides table and row shading;
css	(HTML only) Adds information to the cell-format CSS code
rtf	(RTF only) Adds information to the cell-format RTF code
start	Inserts content at the top of the cell
end	Adds content at the bottom of the cell

Some examples:

```
[Cell]
; These are the default RTF cell properties:
margin left = 6pt
margin right = 6pt
margin top = 6pt
margin bottom = 6pt
border left = LightBorder
border right = LightBorder

[EmptyCell]
border left = NoBorder
border top = NoBorder

[BodyCell]
based = Cell

[ShadedCell]
based = BodyCell
shading = LightGreyShade

[XtraShadedCell]
based = BodyCell
shading = DarkGreyShade

[RowHeadCell]
based = BodyCell
```

```

shading = LightGreyShade
border right = HeavyBorder

[ColHeadCell]
based = BodyCell
shading = LightGreyShade
border bottom = HeavyBorder

```

## 7.8 Configuring page layouts for RTF output

Define page layouts in a page-format template file referenced from your project configuration file (or from another format template) as follows:

```

[Templates]
; Pages = path to file containing properties for page layouts.
Pages = path/to/mypagelayouts.ini

```

See §7.2 [Working with format configuration files](#) on page 110.

*In this section:*

§7.8.1 [Establishing default properties for output pages](#) on page 134

§7.8.2 [Configuring output section formats](#) on page 135

§7.8.3 [Configuring page header and footer formats](#) on page 136

§7.8.4 [Forcing page and column breaks](#) on page 137

§7.8.5 [Configuring border formats for output pages](#) on page 137

§7.8.6 [Configuring a title page \(an example\)](#) on page 138

### 7.8.1 Establishing default properties for output pages

Document properties include the following:

two sided	Yes (Word “facing pages”) or No (single sided)
start right	Yes if the first two-sided page is always a right page
start section	Section format name for the starting section

These are the default document properties:

```

[Document]
category = Pages
help = This is the default page layout for RTF output
two sided = yes
start right = yes
start section = Section

```

Document properties include defaults for most section properties; see §7.8.2 [Configuring output section formats](#) on page 135:

```

[Document]
; defaults for sections:
mirror margins = yes
page number type = Num
page number start = 1
orient = portrait
page wide = 8.5in
page high = 11in
margin left = 1in
margin right = 0.75in
margin top = 1in
margin bottom = 0.75in
gutter = 0
column count = 1

```

```

column gap = 0
column rule = no
border top = None
border bottom = None
border left = None
border right = None
border header = yes
border footer = yes
header top = 0.25in
footer bottom = 0.25in

```

## 7.8.2 Configuring output section formats

The names of section formats must end in `Section`. The default section format is named `Section`. [Table 7-8](#) lists the properties you can define for sections.

**Table 7-8 RTF output section properties**

Section property	Valid property values
help	Any text to describe purpose or use of format
start side	right, left, or page (default); indicates which page type starts the section; page means the section starts on the next page, regardless of page type; and if <code>first page=Yes</code> , applies first-page headers and footers, regardless of page type
mirror margins	Yes or No; if Yes, and if document property <code>two sided</code> is Yes, left and right margins of each left page are exchanged, so that inner margins of both left and right pages are equal to the right page's left margin, and outer margins of both pages are equal to the right page's right margin
page number type	Num (default), UAlpha, LAlpha, UCRom, or LCRom; see <a href="#">Table 8-2</a> on page 147
page number start	First value to use, a digit regardless of type; 1 could indicate 1, A, a, I, or i, depending on the value of <code>page number type</code> .
orient	portrait (default) or landscape
page wide page high	Any CSS units for the physical page dimensions, such as in or mm.
margin left margin right margin top margin bottom	Distances from the edge of the paper to the edge of the body area (which may have multiple columns), ignoring any header or footer
first space above	Distance from the normal top margin of the body area to the top margin for the first page
gutter	Width; if document property <code>two sided=Yes</code> , adds to the right page's left margin and the left page's right margin; if not two sided, adds to the left margin of every page;
column count	Number of columns; if more than one, columns are snaking, top to bottom, left to right
column rule	Yes or No (default); if Yes, and <code>column count &gt; 1</code> , adds a single thin black line between columns, that is just long enough to reach the bottom of the longest column content (not the bottom of the body area)
border left border right border top border bottom	Names of border formats that surround the body area (see <a href="#">§8.2 Defining border format components</a> on page 144) or None
border header border footer	Yes or No; if Yes, the page border also surrounds the header and footer, not just the body area
header top	Distance from top edge of page to top of header

**Table 7-8 RTF output section properties (continued)**

Section property	Valid property values
footer bottom	Distance from bottom edge to bottom of footer
first page	Yes or No; if Yes, use the first-page header and footer, if any
header first footer first	Names of header and footer formats for first pages (see §7.8.3 <a href="#">Configuring page header and footer formats</a> on page 136) or None; used only if first page=Yes
header left footer left	Names of header and footer formats for even pages, if document property two sided =Yes, or None
header right footer right	Names of header and footer formats for odd pages, if document property two sided =Yes, or for all pages if two sided =No, or None

These are the default section properties:

```
[Section]
start side = right
mirror margins = yes
page number type = Num
page number start = 1
orient = portrait
page wide = 8.5in
page high = 11in
margin left = 1in
margin right = 0.75in
margin top = 1in
margin bottom = 0.75in
gutter = 0
column count = 1
column gap = 0
column rule = no
border top = None
border bottom = None
border left = None
border right = None
border header = yes
border footer = yes
header top = 0.25in
footer bottom = 0.25in
first page = yes
first space above = 1in
header first = FirstHeader
footer first = FirstFooter
header left = LeftHeader
footer left = LeftFooter
header right = Header
footer right = Footer
```

### 7.8.3 Configuring page header and footer formats

The names of header formats must end in Header, and the names of footer formats must end in Footer. The default header and footer formats are named Header and Footer.

The properties you can specify for header and footer formats are as follows:

format	Name of a block format defined in the formats file; see §7.6.5 <a href="#">Specifying inline properties for paragraph and character formats</a> on page 123. Specify any desired borders and shading in the format definition.
--------	--

`content` Can contain text and macros; follows the same rules as for macro content (such as start and end) in other formats. Macros must be defined in a macro library, or in a configuration file, not in a formats file.

To define header and footer formats:

```
[Header]
help = Page header for RTF output
format = Header
content = <$$_basename> \\tab <$_Styleref(Heading2)>

[Footer]
help = Page footer for RTF output
format = Footer
content = \\chdate \\tab \\chpgn

[LeftHeader]
format = Header
content = <$_Styleref(Heading2)> \\tab <$$_basename>

[LeftFooter]
format = Footer
content = \\chpgn \\tab \\chdate

[FirstHeader]
format = Header
content = \~

[FirstFooter]
format = Footer
content= Copyright \\'A9 2009 Your Company \\tab \\chpgn
```

## 7.8.4 Forcing page and column breaks

To force a page or column break at a place in your DITA document where one would not occur based on topic boundaries or configuration settings, you can insert a PI marker, one of the following:

```
<?dtrtf Break="column" ?>
<?dtrtf Break="page" ?>
```

*See also:*

§7.9 [Inserting line, column, and page breaks in output](#) on page 138

§38.1 [Understanding DITA2Go PI markers](#) on page 717.

## 7.8.5 Configuring border formats for output pages

Border formats defined in a pages file apply to page borders only. Border format names should end in `Border` as a convention. The default border format is no border at all. The properties you can specify are the same as those available for border formats you can define in the formats file; see §8.2 [Defining border format components](#) on page 144.

To specify properties for page borders:

```
[LightBorder]
type = single
color = black
thick = 1pt
space = 6pt

[HeavyBorder]
based = LightBorder
thick = 2pt
```

```
[DoubleBorder]
based = LightBorder
type = double
```

## 7.8.6 Configuring a title page (an example)

These are macros to be invoked from the format properties; the macros would be defined in a macro library, not in the formats file:

```
[AuthorPub]
; this is for the title page below the mainbooktitle
<$_style(Heading2)> \\qc <$Author> \\par \n
<$_style(Heading3)> \\qc <$URL> \\par \n
<$_style(Heading2)> \\sb3600 \\qc <$Converter> \\par \n
<$_style(Heading3)> \\qc <$Date> \\par

[Author]
DITA Test Suite Project, Sourceforge

[URL]
<https://sourceforge.net/projects/ditatestsuite/>

[Converter]
RTF Produced by DITA2Go\\'99\~

[Date]
\\chdpl\~
```

Macro AuthorPub is invoked from the document properties:

```
[BookTitle]
based = Heading1
keep = none
section = TitleSection
text align = center
font size = 36pt
margin bottom = 1in
start = \\line \\line \\line \\line \~
after = <$AuthorPub>
```

Document properties use this section:

```
[TitleSection]
based = Section
vertical align = middle

border top = DoubleBorder
border bottom = DoubleBorder
border left = HeavyBorder
border right = HeavyBorder
border header = no
border footer = no

header first = None
footer first = None
header left = None
footer left = None
header right = None
footer right = None
```

## 7.9 Inserting line, column, and page breaks in output

You can insert line breaks with the following processing instructions:

```
<?dthtm Break="line" ?>    for HTML output
```



`<?dtrtf Break="line" ?>` for RTF output

Unlike adding `<br />`, using a PI for a line break in HTML output will not make your DITA document invalid.

For RTF output, you can also insert column and page breaks with PIs:

`<?dtrtf Break="column" ?>`

`<?dtrtf Break="page" ?>`

See §7.8.4 [Forcing page and column breaks](#) on page 137.



# 8 Configuring format components

---

This section shows how to define components of output formats: numbering, borders, shading, and cross references; and also how to customize headings and labels, and supply values for output text strings. Topics include:

- §8.1 [Managing format components](#) on page 141
- §8.2 [Defining border format components](#) on page 144
- §8.3 [Defining shading format components](#) on page 145
- §8.4 [Overriding border and shading properties](#) on page 145
- §8.5 [Configuring output numbering properties](#) on page 146
- §8.6 [Configuring run-in headings for text formats](#) on page 153
- §8.7 [Defining cross-reference output formats](#) on page 155
- §8.8 [Configuring trademark formats](#) on page 157
- §8.9 [Localizing output headings, labels, and names](#) on page 157

## 8.1 Managing format components

**DITA2Go** uses format components as building blocks for numbering, cross references, borders, and shading. These building blocks are referenced by paragraph, character, table, row, cell, section, or page format definitions in their respective format configuration files.

*In this section:*

- §8.1.1 [Understanding where to define format components](#) on page 141
- §8.1.2 [Basing format component properties on other components](#) on page 142
- §8.1.3 [Assigning additional names to format components](#) on page 142
- §8.1.4 [Understanding format-component name lookups](#) on page 142
- §8.1.5 [Including typographic tags and character formats](#) on page 143

### 8.1.1 Understanding where to define format components

Unlike other format types, you can reference format components even if they are not defined in the same format-file chain as the referencing format. For example, if you reference `LightBorder` in a table format, but `LightBorder` is defined only in your page-layout file, **DITA2Go** can find the definition. However, any definition in the file that contains the reference overrides definitions of the same format component in other format configuration files.

You define format components in a format configuration file referenced either from another format template file or from your project configuration file; see §7.2 [Working with format configuration files](#) on page 110.

To reference a format component configuration file:

```
[Templates]
; Subformats = path to a file where format components are defined
Subformats=%omsyshome%\d2g\formats\d2g_subformats.ini
```

`Subformats` specifies a path to a general format component file, for use when a format component is missing in the present format or general configuration file. If a format component is defined in both the referenced and the referencing file, **DITA2Go** uses the definition in the referencing file; the definitions are not merged.

Associated with certain format components are labels, headings, and other fixed text components. The default settings for these text values are located in language configuration files; see §8.9 [Localizing output headings, labels, and names](#) on page 157. You can override the default values in your project configuration file, or anywhere in the chain of general configuration templates; but not in a format component or format configuration file.

## 8.1.2 Basing format component properties on other components

All format components can have a `based` property, which includes the properties of a referenced format component of the same type. Basing properties works the same way for format components as for formats; see §7.4.5 [Basing format properties on other formats](#) on page 116.

## 8.1.3 Assigning additional names to format components

To map additional format component names to existing format component definitions (for example):

```
[FormatAliases]
; format component name = another format component name
DividerBorder = ThickBorder
```

You can use this section to specify properties for the format component named on the left as though `[FirstSubFormat]based=SecondSubFormat` (see §7.4.5 [Basing format properties on other formats](#) on page 116). Make sure that `SecondSubFormat` is defined somewhere in the format component template chain.

When **DITA2Go** looks up format component definitions, **DITA2Go** replaces each format component name listed on the left with the name assigned to it on the right. This process is not recursive; if you assign format component names `a=b` and `b=c`, when **DITA2Go** needs a definition for `a`, instead **DITA2Go** looks for a definition of `b`, not `c`.

## 8.1.4 Understanding format-component name lookups

If **DITA2Go** fails to find a definition for an assigned format component name in the current format component template file, **DITA2Go** looks for the definition in any configuration files assigned in `[Templates]` to `Formats`, `Tables`, `Pages`, and `Subformats`, in that order.

For example, if a border format is based, **DITA2Go** looks for the base definition using the same process, but starting with the file that contains the sought format component definition, not the original file. For example:

In `formats.ini`:

```
[MyPara]
border bottom = MyBorder

[OtherBorder]
color = blue
```

In `tables.ini`:

```
[MyBorder]
based = OtherBorder

[OtherBorder]
color = red
```

The border under each *MyPara* paragraph will be red, not blue.

### 8.1.5 Including typographic tags and character formats

Many configuration settings allow you to specify text values that will appear in output. You can surround the text with any character format (including formats not defined elsewhere; **DITA2Go** defines them), or with one or more of the following typographic tags:

<code>b</code>	Bold
<code>i</code>	Italic
<code>q</code>	Quote
<code>u</code>	Underline
<code>tt</code>	Monospace
<code>sup</code>	Superscript
<code>sub</code>	Subscript

You can use typographic tags in the definitions of format components, and also in settings in the `Languages` configuration files. Most text-value configuration settings are included in each of the language configuration files, where they have default values appropriate to the language in question; see §8.9 [Localizing output headings, labels, and names](#) on page 157.

And you can nest tags. For example:

```
[StopHead]
form = <b><i>Stop!</i></b>
```

**DITA2Go** automatically terminates any typographic tag at the end of the text, unless you terminate it earlier with a closing tag.

You can also define a character format and use its name in the definition of a format component. For example, in a `Formats` configuration file:

```
[Attention]
based = char
font weight = bold
font style = italic
```

And in a `Subformats` configuration file:

```
[StopHead]
form = <Attention>Stop!</Attention>
```

You can mix format tags and typographic tags in the same format component definition; however, best practice is to assign the typographic tags to the format itself.

**Note:** Do not try to assign more than one `<format>` tag to a format component.

**Quote styles** To specify quote styles for the `<q>` typographic element, in a language configuration file:

```
[ElementText]
; Quotes = left double, right double, left single, and right single,
; in a space-delimited list, for <q> element, nested alternately.
Quotes= " " ` `
```

**DITA2Go** recognizes either the Windows code page or the UTF-8 characters. Quote characters are inserted literally in the encoding specified for the output type. Default values for quote characters are provided in each of the language configuration files; see §8.5 [Configuring output numbering properties](#) on page 146. The values shown here are the default values specified in `d2g_lang_en.ini`. You can override the default values in a general configuration file.

## 8.2 Defining border format components

Border format components apply to paragraph, table, row, and cell formats. When you want borders around text, parts of a table, or components of a page (RTF only), you assign the name of a border component to the item to be bordered. The default is no border.

As a convention, the name of any border component should end in `Border`. If you give a border format component a name that does not end in `Border`, if the definition is in a file different from the file where it is referenced, **DITA2Go** will not be able to find the format component. (But see §8.9 [Localizing output headings, labels, and names](#) on page 157.) [Table 8-1](#) lists the border properties and the values you can assign to them.

**Table 8-1 Properties of border format components**

Property	Description
help	Any text to describe purpose or use of format
type	Names in parentheses are synonyms; hidden overrides all other properties: single (solid) double dot (dotted) dash (dashed) hidden (HTML only) transparent (HTML only) inset (HTML only) outset (HTML only) ridge (HTML only) groove (RTF only) shadow (RTF only) hairline
color	CSS color name; for HTML, this is the fill color in CSS terms
thick	Thickness of border; number and units
space	(RTF only) distance from content

Border format component properties can be based on the properties of other border format components. For example:

```
[LightBorder]
type = single
color = black
thick = 1pt
space = 6pt

[HeavyBorder]
based = LightBorder
thick = 2pt

[DoubleBorder]
based = LightBorder
type = double
```

To override border properties for a particular instance of an element, you can specify a different property in the `outputclass` attribute for that instance. See §8.4 [Overriding border and shading properties](#) on page 145.

## 8.3 Defining shading format components

Shading format components apply to paragraph, character, table, row, and cell formats. When you want shading applied to text, to parts of a table, or to components of a page (RTF only), assign the name of a shading format component to the item to be shaded. The default is no shading.

As a convention, the name of any shading format component should end in `Shade`. If you give a shading format component a name that does not end in `Shade`, if the definition is in a file different from the file where it is referenced, **DITA2Go** will not be able to find the format component. (But see §8.9 [Localizing output headings, labels, and names](#) on page 157.)

Shading properties are as follows:

<code>help</code>	Any text to describe purpose or use of format
<code>type</code>	(RTF only) <code>horiz</code> , <code>vert</code> , <code>fdiag</code> , <code>bdiag</code> , <code>cross</code> , <code>dcross</code> , or <code>dk</code> followed by any of the other patterns (as in <code>dkcross</code> )
<code>color</code>	Color of the pattern, or of the fill if no <code>type</code> specified
<code>background color</code>	(RTF only) pattern background, ignored if no <code>type</code> specified
<code>tint</code>	Percent of <code>color</code> if no pattern, ignored if there is a pattern

Properties `type` and `background color` apply only to RTF output.

For example:

```
[DangerShade]
type = dcross
color = orange

[CautionShade]
color = yellow
tint = 80%

[WarningShade]
color = red
tint = 10%
```

To override shading properties for a particular instance of an element, you can specify a different property in the `outputclass` attribute for that instance. See §8.4 [Overriding border and shading properties](#) on page 145.

## 8.4 Overriding border and shading properties

By default, **DITA2Go** checks the `outputclass` attribute of each element for a format name, and also for possible border and shading format component names; see §6.2

[Specifying options for naming formats](#) on page 87:

```
[ElementOptions]
; UseOutputClassForFormat = Yes (default, use whenever present) or
; No (go on to mappings in [*FormatMaps] as the next choice)
UseOutputClassForFormat = Yes
; OutputclassHasBorderShadeFormats = Yes (default, look for border and
; shading format specifications in outputclass attributes), or No.
OutputclassHasBorderShadeFormats = Yes
```

To override border or shading properties for a particular instance of an element, you can specify different properties in the `outputclass` attribute for that instance.

When `UseOutputClassForFormat=Yes` and `OutputclassHasBorderShadeFormats=Yes`, **DITA2Go** checks `outputclass` attributes for multiple classes and separates them. If there are multiple classes, **DITA2Go** uses the first as the real `outputclass` (usually a block or inline format), the second as the border format, and the third as the shading format.

As an alternative, you can insert a **BorderType** or **ShadeType** PI marker in the element; see §38.1 [Understanding DITA2Go PI markers](#) on page 717.

## 8.5 Configuring output numbering properties

Some forms of output require more than simply mapping an element to an output format or style. For example, ordered lists, footnotes, and headings typically need series numerals or symbols, and possibly prefixes and punctuation, in addition to text content. **DITA2Go** provides numbering streams and assorted building blocks you can define for each output format, to achieve the desired presentational effect.

*In this section:*

- §8.5.1 [Understanding numbering properties](#) on page 146
- §8.5.2 [Defining number streams](#) on page 147
- §8.5.3 [Defining number format components](#) on page 148
- §8.5.4 [Defining footnote numbering](#) on page 150
- §8.5.5 [Considering examples of numbering schemes](#) on page 150

*See also:*

- §8.6 [Configuring run-in headings for text formats](#) on page 153

### 8.5.1 Understanding numbering properties

Your document might include more than one type of element whose output format should be numbered sequentially. Ordered lists are one example; footnotes are another. Chapters, figures, and tables might also be numbered.

*Streams and counters*

If several sequential-number series depend on other series (such as figure or table numbering restarting with each chapter), each such series requires a separate *counter* in the same numbering *stream*. For example, the numbers on ordered lists might form one stream. Numbers on chapter, figure, and table titles might each need a counter in a different stream, numbered independently of the ordered-list stream.

To organize streams and their counters:

1. Name each number stream, and assign properties to its counters; see §8.5.2 [Defining number streams](#) on page 147.
2. Define number format components (see §8.5.3 [Defining number format components](#) on page 148) in terms of streams and counters.
3. Assign a number format component to each paragraph format that needs numbering; see §7.6.6 [Specifying block properties for paragraph formats](#) on page 124.

*Keep a counter from resetting*

You can specify whether other counters should be left alone when a given counter is incremented. For example, if *FigureTitle* has counter 2 and *TableTitle* has counter 3, and both are in the same stream, you would not want each instance of *FigureTitle* to reset the *TableTitle* counter. To prevent unwanted resets, you assign a *keep* property to the counters that should be left alone for each format.



<i>Restart stream numbering</i>	To restart numbering for ordered lists, you must assign a different format to the first item in the list, so you have a way to trigger the restart. For example, assign <i>NumberedFirst</i> to each first item, and <i>Numbered</i> to all the other items; see §6.4 <a href="#">Mapping element paths to output formats</a> on page 91. Both formats must be in the same stream. To force every instance of <i>NumberedFirst</i> to begin the numbering over again, you assign a <i>Start</i> property to <i>NumberedFirst</i> .
<i>Cross references to numbered formats</i>	For cross references to items in numbered formats, you can specify whether the cross reference should include a prefix (such as “Chapter”) before the number, a suffix (such as a period) after the number, and maybe a tab (for vertical alignment in RTF) or a space (for HTML). By default, <b>DITA2Go</b> includes only the number itself in cross references to a numbered item. You can add embellishments such as a prefix, a suffix, and tabs. See §8.7 <a href="#">Defining cross-reference output formats</a> on page 155.

## 8.5.2 Defining number streams

You use number streams to coordinate numbering for those elements whose occurrences in your document should be numbered sequentially, or should appear with a constant prefix or suffix (or both). As a convention, the name of any number stream should end in *Stream*; the **DITA2Go** Configuration Manager relies on this convention.

These are the default stream names defined in format components configuration templates:

PartStream	Part numbers, like volume numbers
ChapterStream	Chapter title, all numbered subheads in chapter
AppendixStream	Appendix title, all numbered subheads in appendix
ListStream	Ordered lists, and ordered lists nested within them
FootnoteStream	Footnotes referenced in text
TblFootnoteStream	Footnotes referenced in table cells

You can add names for other number streams of your own, and you can change the default names. However, if you change a default name, you must change it in every place the name is referenced.

If you add your own number streams, you must define their properties. [Table 8-2](#) lists the available properties and the values you can assign to those properties.

**Table 8-2 Properties of number streams**

Property	Default	Description	
id	x	( <i>RTF only</i> ) Stream ID to use in Word SEQ fields, preferably a single alphabetic character	
counters	1	Maximum number of counters available in stream	
counter types	Num	Space-delimited list of types, one for each counter, from the following list:	
		Num	Arabic numeral
		UCRom	Uppercase Roman numeral
		LCRom	Lowercase Roman numeral
		UCAAlpha	Uppercase letter
		LCAAlpha	Lowercase letter
		Sym	Symbol

**Table 8-2 Properties of number streams**

Property	Default	Description
repeat	last	Which counter types to repeat, if there are more counters than counter types:
		last Repeat only the last counter type
		all Repeat the entire sequence of types, from the beginning
reset	none	When to restart footnote numbering:
		topic <i>(HTML only)</i> At the start of each topic, for all footnotes <i>(RTF only)</i> At each page boundary, for all footnotes
		table At the start of each table, for table footnotes
		none Footnote numbers continue through entire document; for example, for endnotes
symbols	*	Space-delimited list of symbols to use when counter type is Sym

The value for `symbols` can consist of more than one character. If more are needed than are listed here, they continue with an increasing number of the last symbol listed. So if you use the default, a single asterisk, the sequence is \*, \*\*, \*\*\*, \*\*\*\*, and so forth. Symbols can be `<u+NNNN/>` for a Unicode character, where `NNNN` is the hexadecimal code point, as in `<u+2020/>` for a dagger.

For example (these are default definitions):

```
[ChapterStream]
help = Number stream for chapters
id = C
counters = 10
counter types = Num

[AppendixStream]
help = Number stream for appendixes
id = A
counters = 10
counter types = UCAAlpha Num
repeat = last

[List1Stream]
help = Number stream for tier 1 ordered lists
id = L
counters = 3
counter types = Num LCAAlpha LCRom
```

### 8.5.3 Defining number format components

To apply number streams, numeric or alphabetic counters, prefixes, suffixes, and other repeating properties to output formats, you must define number format components in terms of these properties. As a convention, the name of any number format component should end in `Num`. [Table 8-3](#) lists the properties and the values you can assign to number format components.

**Table 8-3 Properties of number format components**

Property	Description
stream	Name of number stream to use (see §8.5.2 <a href="#">Defining number streams</a> on page 147), or <code>none</code> for no stream (for example, for bulleted lists)
counter	Numeric ID of counter to increment, 0 to not increment any counters

**Table 8-3 Properties of number format components**

Property	Description
position	Where to display the number in relation to the text: start           As a prefix to the text (default) end            As a suffix to the text (usually for equations)
start	Value to restart counter with (usually 0 or 1), default is to continue numbering
keep	Space-delimited list of counters to leave alone; default is to reset all higher-numbered counters in the stream
name	Name to use in <code>&lt;name/&gt;</code> token, if not defined in <code>[NumberFormatsText]</code> in the language file; see §8.5 <a href="#">Configuring output numbering properties</a> on page 146
form	Text plus any or all of the following building blocks: <code>&lt;name/&gt;</code> For the text name associated with the format in <code>[NumberFormatsText]</code> in a language file; see §39.1.6 <a href="#">Understanding how language templates are organized</a> on page 730 <code>&lt;counter1/&gt;</code> ,   To Identify which of the counters to display <code>&lt;counter2/&gt;</code> , ... <code>&lt;tab/&gt;</code> (RTF only) Each tab advances by the amount in <code>[WordOptions]AnumTabWidth</code> (see §15.3.3 <a href="#">Converting autonumbered formats</a> on page 226) (HTML only) Adds a space <code>&lt;spc/&gt;</code> For a space, used to preserve trailing spaces; for HTML, becomes <code>&amp;nbsp;</code> <code>&lt;format&gt;</code> For a character format, reset at the end of the autonumber or earlier by a <code>&lt;/format&gt;</code> ; typographic tags are valid; see §8.1.5 <a href="#">Including typographic tags and character formats</a> on page 143 <code>&lt;u+NNNN/&gt;</code> For a Unicode character, where <code>NNNN</code> is the hexadecimal code point, as in <code>&lt;u+2020/&gt;</code> for a dagger

The `<name/>` building block is for a text label that becomes part of the output number string. Text labels are defined in each of the language configuration files; see §8.9 [Localizing output headings, labels, and names](#) on page 157. For example, in `d2g_lang_en.ini`, the default labels are as follows:

```
[NumberFormatsText]
; number format name = text to use in numbering string for <name/>
ChapterNum = Chapter
TableNum = Table
FigureNum = Figure
EquationNum = Equation
```

If you use different format names, the default text values do not apply, and you must specify those values explicitly. Or change the format names in the language configuration file; see §8.9 [Localizing output headings, labels, and names](#) on page 157.

You can mix format tags and typographic tags in the same format component definition; however, best practice is to assign the typographic tags to the format itself. See §8.1.5 [Including typographic tags and character formats](#) on page 143.

**Note:** Do not try to assign more than one `<format>` tag to a format component.

## 8.5.4 Defining footnote numbering

Footnote numbering can use any of the `counter` type values listed in [Table 8-2](#) on page 147 (default `Num`), followed by a `reset` value of `None` or `Topic` (default `Topic`), to specify what restarts the footnote numbering stream.

By default, text footnotes are in one stream, and table footnotes are in another stream. If you want table footnotes numbered in the same stream as text footnotes, change the stream named in both `[TblFootnoteNum]` and `[TblFootnoteRefNum]`, in format configuration template `%omshome%\d2g\formats\d2g_subformats.ini`; or override the stream name in your own version of these number format components.

## 8.5.5 Considering examples of numbering schemes

*In this section:*

- §8.5.5.1 [Numbering ordered lists](#) on page 150
- §8.5.5.2 [Modifying the appearance of numbers](#) on page 151
- §8.5.5.3 [Numbering chapters, figures, and tables](#) on page 152
- §8.5.5.4 [Specifying symbols for bulleted lists](#) on page 152

### 8.5.5.1 Numbering ordered lists

The following settings handle the items in two-level ordered lists.

Paragraph format definitions:

```
[Numbered]
number = ListNum
xref = NumXref

[NumberedFirst]
based = Numbered
number = ListFirstNum

[Numbered2]
based = Numbered
number = List2Num

[Numbered2First]
based = Numbered2
number = List2FirstNum
```

Number format component definitions:

```
[ListStream]
counters = 2
counter types = Num LCAalpha

[ListNum]
stream = ListStream
counter = 1
form = <counter1/>.<tab/>

[ListFirstNum]
based = ListNum
start = 1

[List2Num]
stream = ListStream
counter = 2
form = <tab/><counter2/>.<tab/>
```

```
[List2FirstNum]
based = List2Num
start = 1
```

In this example, format *NumberedFirst* starts a list at number 1, using Arabic numerals, and resets the following (higher-numbered) counter; format *Numbered* continues the same list. Format *Numbered2* starts or continues a subordinate list, using lowercase letters instead of numerals.

**Note:** For ordered lists, these settings take effect in HTML only if you do *not* specify HTML list styles; see §30.11.2 [Converting list formats to HTML list styles](#) on page 585.

### 8.5.5.2 Modifying the appearance of numbers

Suppose that for RTF output you want to make numbers on procedure steps and substeps bold, and give them a color different from the text; for example, RGB 42-96-248. To do this, in `local_d2rtf_formats.ini` you would modify the inline format used in the numbering formats for steps and substeps.

These are the format definitions in system `d2rtf_formats.ini`:

```
[StepNumbered]
based=Numbered1
xref=StepXref

[StepNumberedFirst]
based=Numbered1First
xref=StepXref
```

Look at the format on which they are based:

```
[Numbered1]
based=Body
margin left=0.25in
text indent=-0.25in
margin top=2pt
margin bottom=2pt
tabs=0.25in
number=List1Num
xref=NumXref
```

You see that the number format is `List1Num`. That is defined (in system `d2rtf_subformats.ini`) as:

```
[List1Num]
stream = List1Stream
counter = 1
form = <counter1/>.<tab/>
```

You need to override the `form` value. Add a section for the inline format in `local_d2rtf_formats.ini`, specifying a different number format; for example:

```
[StepNumbered]
number=MyList1Num
```

and define that number format in `local_d2rtf_subformats.ini`:

```
[MyList1Num]
based=List1Num
form=<BlueBold><counter1/>.<tab/>

[BlueBold]
color=rgb(42,96,148)
based=Bold
```

Now the numbers on steps and substeps will be bold and blue in RTF output.

### 8.5.5.3 Numbering chapters, figures, and tables

The following settings configure the numbering properties of chapter, figure, and table titles.

Paragraph format definitions:

```
[ChapterTitle]
number = ChapterNum

[FigureTitle]
number = FigureNum

[TableTitle]
number = TableNum
```

Number format component definitions:

```
[ChapterStream]
counters = 8
counter types = Num

[ChapterNum]
stream = ChapterStream
counter = 1
position = start
form = <name/> <counter1/>.<tab/>

[FigureNum]
stream = ChapterStream
counter = 6
keep = 7 8
form = <i><name/> <counter1/>--<counter6/> <tab/>

[TableNum]
stream = ChapterStream
counter = 7
keep = 8
form = <i><name/> <counter1/>--<counter7/> <tab/>
```

In this example, all three formats are assigned to the same stream, and all three number series use Arabic numerals:

- ChapterNum has counter number 1. Because FigureNum and TableNum have counters with higher numbers, by default the numbering of *FigureTitle* and *TableTitle* will be reset every time a new instance of *ChapterTitle* appears in the output.
- FigureNum has counter number 6. To prevent the numbering of *TableTitle* from being reset every time *FigureTitle* appears, FigureNum assigns the keep property to counter 7.
- TableNum has counter 7. To prevent the numbering of whatever format has counter 8 from being reset every time *TableTitle* appears, TableNum assigns the keep property to counter 8.

### 8.5.5.4 Specifying symbols for bulleted lists

Bullet symbols are specified by their Unicode values for both HTML and RTF output, but the way you express those Unicode values is somewhat different.

*Bullet symbols for  
HTML output*

For example, to specify different bullets for bulleted lists at different levels in HTML, you assign a Unicode hexadecimal value to the `form` property of a format component:

```
[Bullet1Num]
stream = none
form = <U+2022/><tab/>
```

```
[Bullet2Num]
stream = none
form = <U+25E6/><tab/>

[Bullet3Num]
stream = none
form = <U+25AA/><tab/>
```

There is no color property for bullets in CSS. To change the color of the bullet for HTML output, you would have to use a graphic that has the color and shape you want.

*Bullet symbols for  
RTF output*

To change the bullet style for RTF output, you assign a Unicode decimal value to the `start` property of the bulleted format itself (rather than a format component). For example the system definition of *Bulleted1* specifies the following `start` property for RTF:

```
start = {\uc1\u8226*}\tab~
```

This is the RTF code to put a Unicode bullet (or asterisk, if the specified bullet is not in the font), then a tab, then a fixed space. You can change the Unicode number to the number for the bullet style you want (in *decimal*!). For example, for a square bullet, you would use `\u9724*`. The asterisk at the end of the Unicode number is the character to use if the font does not include the designated Unicode glyph.

To change the color of the bullet you must add `\cf` codes, one before the bullet to add the color, and one after it to set the color back to black. To specify the color, you need the Word index number, which you can insert with a predefined macro; for example:

```
<$_colornum("rgb(42,96,148)")>
```

See Table 37-2 [Predefined macros for RTF output](#) on page 684. So the value of the `start` property for your modified bulleted format would be as follows:

```
start={\cf<$colornum("rgb(42,96,148)")>\uc1\u9724*}\tab~
```

See Table 8-3 [Properties of number format components](#) on page 148.

## 8.6 Configuring run-in headings for text formats

You can precede paragraphs or character spans in a given format with predefined text that may have a character format different from that of the paragraph or span itself. This fixed text constitutes a run-in heading. For example:

**Note:** The framble must be glommed before you can dechurf it.

In this example, *Note:* is a run-in heading, with a character format different from that of the main paragraph content. As a convention, the name of any run-in heading should end in `Head`.

Run-in headings are assigned to text formats with property `runin`; see §7.6.6 [Specifying block properties for paragraph formats](#) on page 124. Run-in headings are often used for block elements for which you have specified that the values of one or more element attributes should determine the output format in each case; see §6.5 [Mapping element attributes to output formats](#) on page 95.

Run-in headings have only one property: `form`. Table 8-4 lists the building blocks you can use for the `form` property of a run-in heading.

**Table 8-4 Building blocks for run-in heading formats**

Building block	Description
<code>&lt;name/&gt;</code>	For the text of the heading in [RuninHeadText] in a language file; see §8.9 <a href="#">Localizing output headings, labels, and names</a> on page 157
<code>&lt;tab/&gt;</code>	( <i>RTF only</i> ) Each tab advances by the amount in [WordOptions]AnumTabWidth (see §15.3.3 <a href="#">Converting autonumbered formats</a> on page 226) ( <i>HTML only</i> ) Adds a space
<code>&lt;spc/&gt;</code>	For a space, used to preserve trailing spaces; for HTML, becomes <code>&amp;nbsp;</code>
<code>&lt;format&gt;</code>	For a character format, reset at the end of the run-in heading or earlier by a <code>&lt;/format&gt;</code> ; typographic tags are valid; see §8.1.5 <a href="#">Including typographic tags and character formats</a> on page 143

You can mix format tags and typographic tags in the same run-in heading definition; however, best practice is to assign the typographic tags to the format itself. See §8.1.5 [Including typographic tags and character formats](#) on page 143.

**Note:** Do not assign more than one `<format>` tag to a run-in heading.

For example:

```
[MethodHead]
form = <i>Method:</i><spc/>

[WarningHead]
form = <b><i><name/>!</i></b><br/>
```

In the last example, `<br/>` causes the content of the main paragraph or character span to start on the next line after the heading; otherwise, the run-in heading becomes the start of the first line of content.

To create a hanging indent for a paragraph format that has property `runin`, you might have to fiddle with properties `margin left` and `text indent` until everything lines up the way you want; see §7.6.6 [Specifying block properties for paragraph formats](#) on page 124. However, you will not be able to produce a perfect left margin for the text this way, and you will most likely have to use a different combination for every possible heading content in every supported language.

You can include any fixed text content as part of the `form` property for a run-in heading, as in the first example above. However, if you want to be able to substitute text in other languages, use the `<name/>` building block and create an entry for the run-in heading format in the appropriate language configuration file; see §8.9 [Localizing output headings, labels, and names](#) on page 157.

To specify text for the `<name/>` building block of a run-in heading, in a language configuration file or a general configuration file assign the text to the run-in heading format name:

```
[RuninHeadText]
; use this for runin head format components
NameOfHead = Text of heading
```

For example, in `d2g_lang_de.ini`:

```
[RuninHeadText]
WarnHead = Achtung
```

You can include punctuation with the text, but you cannot use any `<tags>` in language configuration settings.



## 8.7 Defining cross-reference output formats

Cross-reference formats are associated with cross-reference links to different types of text formats in your document. The name of every cross-reference format must end in `Xref`.

The following cross-reference formats are defined in format component configuration files:

<u>Referenced content</u>	<u>Cross-reference format</u>	<u>With page number (RTF only)</u>
Numbered list item	NumXref	NumPageXref
Unnumbered item	TextXref	TextPageXref
Footnote	FtnXref	FtnPageXref
Heading	TitleXref	TitlePageXref
Figure number	FigureXref	FigurePageXref
Table number	TableXref	TablePageXref
Figure title	FigureTitleXref	FigureTitlePageXref
Table title	TableTitleXref	TableTitlePageXref
Equation	EquationXref	EquationPageXref
Step	StepXref	StepPageXref
TOC item	TOCTitleXref	
Topic title, from index (compact)	IndexIconXref	IndexPageXref
Topic title, from index (full)	IndexTitleXref	

These are the default cross-reference formats. You can add other cross-reference formats of your own, and you can change the default names. However, if you change a default name, you must change it in every place that name is referenced; notably in general configuration section `[ElementOptions]`; see §6.3.3 [Mapping cross-reference outputclass attributes to formats](#) on page 91.

You define properties of cross-reference formats by assigning building blocks to `form`, the sole cross-reference format property:

```
[TypeOfXref]
form = <building-block1> <building-block2> ...
```

[Table 8-5](#) lists the building blocks you can assign to property `form` to define a cross-reference format. [Table 8-6](#) shows the default values for the `form` property that are assigned to the default cross-reference format names.

**Table 8-5 Building blocks for cross-reference formats**

<b>Building block</b>	<b>Description</b>
<code>&lt;paratext/&gt;</code>	Paragraph content, usually of the following <code>title</code> element
<code>&lt;autonum/&gt;</code>	Entire autonumber of the referenced item, except for any tabs
<code>&lt;numonly/&gt;</code>	Just the number part of the autonumber, without anything before the first counter or after the last counter
<code>&lt;tab/&gt;</code>	( <i>RTF only</i> ) Each tab advances by the amount in <code>[WordOptions]AnumTabWidth</code> (see §15.3.3 <a href="#">Converting autonumbered formats</a> on page 226) ( <i>HTML only</i> ) Adds a space Any tab present in the referenced autonumber is dropped
<code>&lt;format&gt;</code>	Character format, reset at the end of the cross reference, or earlier by a <code>&lt;/format&gt;</code> ; typographic element names valid; any character format in the referenced autonumber format is dropped <b>Note:</b> if the format is for footnote references, do <i>not</i> close it, or the tag will be ignored

**Table 8-5 Building blocks for cross-reference formats (continued)**

Building block	Description
<u+NNNN/>	Unicode character, where <i>NNNN</i> is the hexadecimal code point, as in <u+2020/> for a dagger
<spc/>	A fixed-width nonbreaking space
<page/>	( <i>RTF only</i> ) The target page number
<tag/>	( <i>HTML only</i> ) The tag to get text from

You can use character format tags and typographic tags around the format definitions; however, do not use tags that have the same names as building blocks: `paratext`, `autonum`, `numonly`, or `tab`. See §8.1.5 [Including typographic tags and character formats](#) on page 143.

**Note:** Do not try to assign more than one `<format>` tag to a cross-reference format component.

Some default cross-reference formats are defined as follows:

```
[TitleXref]
form = <autonum/> <paratext/>

[NumXref]
form = <autonum/>

[FtnXref]
form = <sup><numonly/>

[TextXref]
form = <paratext/>
```

To apply these cross-reference formats to the paragraph formats they reference (see §7.6.6 [Specifying block properties for paragraph formats](#) on page 124):

```
[Heading1]
number = Head1Num
xref = TitleXref

[NumberedFirst]
number = ListFirstNum
xref = NumXref

[Numbered]
number = ListNum
xref = NumXref

[Footnote]
number = FootnoteNum
xref = FtnXref
```

The default cross-reference format names listed in [Table 8-6](#) are used only when no cross-reference format name is specified in the `xref` element `outputclass` attribute.

**Table 8-6 Default cross-reference format names and definitions**

Format name	Cross reference to:	Default format (form property)
TitleXref	Numbered formats; typically includes the title of the referenced item; default for formats assigned to topic, section, fig, and table elements	<autonum/> <paratext/>
NumXref	Items in ordered lists; typically includes only the number; default for formats assigned to li elements	<autonum/>
FtnXref	Footnotes, that typically include only the superscript number; default for formats assigned to fn elements	<sup><numonly/>
FigureXref	Figures, using only the figure number	Figure <numonly/>
TableXref	Tables, using only the table number	Table <numonly/>
FigureTitleXref	Figures, using the figure title	<autonum/> <i><paratext/></i>
TableTitleXref	Tables, using the table title	<autonum/> <i><paratext/></i>
StepXref	Steps	Step <numonly/>
MapTitleXref	TOC items	<autonum/> <paratext/>
TextXref	Text elements without numbers or titles; default for anything else	<paratext/>

## 8.8 Configuring trademark formats

**DITA2Go** provides three default format components for the <tm> element, based on the value of the tmttype attribute:

```
[TradeMark]
; tmttype=tm
form = <U+2122/>

[RegMark]
; tmttype=reg
form = <sup><U+00AE/></sup>

[ServiceMark]
; tmttype=service
form = <U+2120/>
```

You can edit these definitions. If you include a <name/> building block, **DITA2Go** looks up the name in section [MarkText] in a language configuration file; see §8.9 [Localizing output headings, labels, and names](#) on page 157. No other settings are required.

## 8.9 Localizing output headings, labels, and names

Your **DITA2Go** distribution includes several configuration templates with names of the form d2g\_lang\_LL.ini. These language configuration templates contain settings that specify the content of text items such as headings, labels, and file names that will appear in output. Collecting these settings in separate configuration templates gives you a convenient place to localize all such values for your project.

*In this section:*

§8.9.1 [Specifying an output language for your project](#) on page 158

§8.9.2 [Overriding language settings](#) on page 158

§8.9.3 [Specifying language-specific text values](#) on page 158

## 8.9.1 Specifying an output language for your project

To specify which language configuration template to use for your **DITA2Go** project, in your project configuration file:

```
[Templates]
; Languages = path to language-specific .ini file
Languages = %OMSYSHOME%\d2g\local\lang\local_d2g_lang_en.ini
```

**DITA2Go** checks the referenced language template whenever a text setting does not appear in your project configuration file or in a document-specific configuration file.

The default language is English, and the default language template is `d2g_lang_en.ini`. To use a different language, reference one of the other language configuration files in `\d2g\local\lang`, or insert a language configuration file of your own in the template chain. See §39.1.6 [Understanding how language templates are organized](#) on page 730.

## 8.9.2 Overriding language settings

You can override the `[Templates]Languages` setting in your project configuration file or in a document-specific configuration file by specifying a different value of `@xml:lang` for an element. If `@xml:lang` calls for a language different from the one the configuration setting specifies, **DITA2Go** switches to the indicated language file for that element and its children.

Omni Systems supplies language files for English, French, Spanish, German, Czech, and Russian. You can make your own language files for whatever languages you want, and reference them, chained together, from your project configuration file or from a document-specific configuration file. See §39.1.6 [Understanding how language templates are organized](#) on page 730.

To generate text within the content of an element where `@xml:lang=""`, **DITA2Go** uses the default language, rather than omit the text. If you really want the text omitted, use your own language (for example, `enx`), and make a language configuration file for it with empty values for the text strings.

## 8.9.3 Specifying language-specific text values

All configuration sections in **DITA2Go** language files have names that end in `Text`. This suffix is required for reference by the Configuration Manager.

*Text settings  
depend on format  
component  
names*

Many of the settings in language configuration files assign text to the name of a format component. If you use format component names that are different from those specified in the language configuration templates, you must assign each of your preferred names to the corresponding **DITA2Go** format component name; see §8.1.3 [Assigning additional names to format components](#) on page 142.

Several sections in the language configuration files provide text for the `<name/>` building block of the format-component `form` property. For example:

```
[RuninHeadText]
NoteHead = Note:
AttentionHead = Attention:
CautionHead = Caution:
```

See §8.6 [Configuring run-in headings for text formats](#) on page 153.

One language configuration section contains typographic and other miscellaneous text assignments:

```
[ElementText]
; Quotes = left double, right double, left single, and right single,
; in a space-delimited list, for <q> element, nested alternately.
Quotes = " " ` '
; CascadeSeparator = text to use between items in <menucascade>
CascadeSeparator = \~|\~
```

Quotes specifies the delimiters to be used around text converted from <q> elements; see §8.1.5 [Including typographic tags and character formats](#) on page 143.

CascadeSeparator specifies the separator character(s) to be inserted between items converted from <uicontrol> elements; the default is “ | ”, a vertical bar with a space on either side. The separator does not appear if the <uicontrol> elements are not in a <menucascade>. However, it does appear for any elements derived from those two.



# 9 Specifying conditional processing

---

**DITA2Go** allows you to filter on any value of any DITA attribute, not just the official props-derived attributes. This section shows how to direct **DITA2Go** to handle conditional processing attributes and other kinds of selective processing in your DITA documents. Topics include:

- §9.1 [Extracting conditions from ditaval files](#) on page 161
- §9.2 [Defining conditional actions](#) on page 162
- §9.3 [Including flags for ditaval conditions](#) on page 165
- §9.4 [Configuring conditional flags](#) on page 166
- §9.5 [Assigning attributes with conditional flags](#) on page 169
- §9.6 [Scoping and filtering within maps](#) on page 169

## 9.1 Extracting conditions from ditaval files

If your **DITA2Go** project includes one or more ditaval files, **DITA2Go** can use the content of these files to modify or annotate conversion output. In other words, if you have an existing project, perhaps using the DITA Open Toolkit, you are not forced to change to **DITA2Go** proprietary settings. As an alternative, **DITA2Go** provides configuration settings that replace and extend the functionality of ditaval files; see §9.2 [Defining conditional actions](#) on page 162.

If you do not have an existing project, you can go either way, depending on familiarity and specific project needs. If you are still using the Open Toolkit to generate PDF output, for example, you might need ditaval files for that, and you may find it easier to tweak a ditaval file for HTML than to redo its provisions the **DITA2Go** way.

*In this section:*

- §9.1.1 [Specifying a single ditaval file](#) on page 161
- §9.1.2 [Including wildcards in ditaval statements](#) on page 161
- §9.1.3 [Processing complex otherprops settings](#) on page 162

### 9.1.1 Specifying a single ditaval file

To specify a single .ditaval file that applies to your entire DITA document:

```
[ConditionOptions]
; DitavalFile = path to ditaval file to read.
DitavalFile = d:\path\to\my.ditaval
```

When you specify a value for `DitavalFile` in your project configuration file, for HTML output **DITA2Go** combines each attribute name and value to form a class name for use in the file designated by `CSSFlagsFile` (see §9.3 [Including flags for ditaval conditions](#) on page 165); for example, `platformlinux`. This setting is ignored for RTF output.

If your project configuration file does not specify a value for `DitavalFile`, instead **DITA2Go** uses values specified in the `[Conditional*]` sections described in §9.3 [Including flags for ditaval conditions](#) on page 165.

### 9.1.2 Including wildcards in ditaval statements

You can use wildcards `?` and `*` in attribute `@val` settings in `<prop>`. For example:

```
<prop att="platform" val="win*" action="exclude" />
```

to exclude elephants where `platform="win2k winxp win7"` in one statement instead of three. This ensures that when someone adds win8 later, that too will be excluded.

### 9.1.3 Processing complex otherprops settings

If your DITA files contain complex otherprops values, you might want **DITA2Go** to include these values in output. The default is to ignore complex otherprops values:

```
[ConditionOptions]
; ComplexOtherprops = No (default, ignore complex settings because
; they were deprecated in DITA 1.1), or Yes
ComplexOtherprops = Yes
```

When `ComplexOtherprops=Yes`, **DITA2Go** processes any complex otherprops values as though the group name is a distinct attribute name. For example:

```
otherprops="proglang(java cpp) commentformat(javadoc html)"
```

**DITA2Go** treats this example as exactly equivalent to:

```
proglang="java cpp" commentformat="javadoc html"
```

and processes it as:

```
proglang="java cpp"
```

and:

```
commentformat="javadoc html"
```

You can also have simple otherprops values:

```
otherprops="whatever"
```

But you cannot mix both in the same attribute:

```
INVALID: otherprops="proglang(java cpp) whatever"
```

For use in ditaval files, otherprops is always simple:

```
<prop att="otherprops" val="whatever" action="exclude">
```

For complex otherprops in ditaval files, you would use:

```
<prop att="proglang" val="cpp" action="exclude">
<prop att="commentformat" val="html" action="exclude">
```

without mentioning they were packaged in otherprops.

Complex otherprops values were “deprecated in favor of attribute specialization” in DITA version 1.1, and are still listed that way in version 1.2; therefore the default value of `ComplexOtherprops` is No.

## 9.2 Defining conditional actions

The settings in this section replace and extend the functionality of ditaval files. However, if conditional processing for your DITA document is specified in one or more ditaval files, **DITA2Go** ignores the settings in this section.

*In this section:*

[§9.2.1 Understanding the syntax of conditional action settings](#) on page 163

[§9.2.2 Flagging content for special treatment in output](#) on page 163

[§9.2.3 Specifying default conditions for inclusion or exclusion](#) on page 163

[§9.2.4 Including or excluding content based on attribute values](#) on page 164

[§9.2.5 Passing attribute values through in output](#) on page 165



## 9.2.1 Understanding the syntax of conditional action settings

Conditional action settings take the form:

```
attribute = value1 value2 ...
```

For each type of conditional action, the *attribute* name in the configuration setting should be one of the following:

```
props
audience
platform
product
otherprops
a specialization of props
```

The *value* list, if there is more than one value, must be space delimited. Values can include wildcards ? and \*; for example:

```
platform = win*
```

instead of:

```
platform = win2k winxp win7
```

## 9.2.2 Flagging content for special treatment in output

To flag certain attribute values for differential treatment of content:

```
[ConditionalFlagging]
; attribute name = list of value=flag pairs.
attrname = value1=flag1 value2=flag2 ...
```

For each value listed, for HTML output the associated flag is used as the class name for inclusion in the CSS flags file (see §9.3 [Including flags for ditaval conditions](#) on page 165), with the properties defined in [ConditionalFlags]; see §9.4 [Configuring conditional flags](#) on page 166.

For example:

```
[ConditionalFlagging]
platform= mac=blueback win=greenback
```

Content with attribute `platform=mac` would be rendered with a blue background, while content with attributed `platform=win` would be rendered with a green background, provided you have defined flags `blueback` and `greenback` to produce this effect; see §9.4 [Configuring conditional flags](#) on page 166.

**Note:** Flag name `conflict` is reserved for disallowed combinations of flag properties; see §9.4 [Configuring conditional flags](#) on page 166.

## 9.2.3 Specifying default conditions for inclusion or exclusion

By default, all content is included in **DITA2Go** output, unless the *action* conditional processing attribute in a ditaval file for the content specifies `exclude`, or a corresponding setting in your configuration file specifies the attribute value(s) required to trigger exclusion.

To specify the default action for all attributes:

```
[ConditionOptions]
; ConditionalDefaults = include (default) or exclude, for all
; attributes unless overridden in the [ConditionalDefaults] section.
ConditionalDefaults = include
```

To specify a different default action for individual attributes (for example):

```
[ConditionalDefaults]
; attribute name = actions to use for any props not set explicitly,
; can be include (default) or exclude.
audience = exclude
```

In this example, all values for audience would be excluded by default.

## 9.2.4 Including or excluding content based on attribute values

For any of the attributes listed in §9.2.1 [Understanding the syntax of conditional action settings](#) on page 163, you can override the default conditional action for specified values, to include or exclude content.

*Overriding  
excluded attribute  
values*

To specify a list of attribute values that should cause an element to be included when the conditional default for the attribute is exclude (see §9.2.3 [Specifying default conditions for inclusion or exclusion](#) on page 163):

```
[ConditionalInclude]
; attribute name = list of values to include, effective when the
; conditional default for the attribute is exclude.
attribute = value1 value2 ...
```

If no value for an attribute on an element is included, the element is excluded; otherwise, it is included.

For example:

```
[ConditionalDefaults]
audience = exclude

[ConditionalInclude]
audience = beginner
```

In this example, of those elements that have an audience attribute, only those with the value beginner would be included, because the default action for audience is to exclude. The result would be output oriented for beginners.

*Overriding  
included attribute  
values*

To specify a list of attribute values that should cause an element to be excluded when the conditional default for the attribute is include (continuing the same example):

```
[ConditionalExclude]
; attribute name = list of values to exclude, effective when the
; conditional default for the attribute is include.
attribute = value1 value2 ...
```

If all values for any attribute on an element are excluded, the element is excluded; otherwise, it is included.

For example:

```
[ConditionalDefaults]
audience = exclude

[ConditionalExclude]
audience = advanced
platform = linux
```

Elements with audience="advanced" would be excluded (redundant, because the default action for audience is to exclude) and so would elements with platform="linux". The result would be output oriented to all but advanced audiences, working on any platform except Linux. Because the default is include for platform, only elements that have the single value linux for platform would be excluded; an element with platform="linux mac" would be included.

## 9.2.5 Passing attribute values through in output

To mark certain attribute values to be passed through so they are available for further processing outside of **DITA2Go**:

```
[ConditionalPassthrough]
; attribute name = list of values to pass through.
attrname = value1 value2 ...
```

Although not rendered as output, attributes with these values are included in the output stream, as possible for the output type. This setting has no effect in RTF output.

For example:

```
[ConditionalPassthrough]
product = Mif2Go DITA2Go
```

Any element with a `product` attribute of `Mif2Go` or `DITA2Go` would be included in output, and the attribute name and value would be attached to the content (but not visible) in whatever form is appropriate for the output type.

## 9.3 Including flags for ditaval conditions

By default, **DITA2Go** includes, in output, flags based on conditional processing attributes in any ditaval files associated with your DITA document, or on settings you specify in your project configuration file.

To omit conditional flags from output:

```
[ConditionOptions]
; UseConditionalFlagging = Yes (default, set flags per ditaval file
; or flag-related sections below), or No (do not include flags)
UseConditionalFlagging = No
```

If `UseConditionalFlagging=Yes` and you are producing HTML output, you might want to include conditional flagging in CSS. **DITA2Go** can flag classes, and optionally create a special CSS file for flagged classes.

To name a special CSS file for flagged classes:

```
[ConditionOptions]
; CSSFlagsFile = name of CSS file to use for flagging classes for
; HTML outputs. If not specified, related settings below ignored.
CSSFlagsFile = flags.css
```

If your project configuration file includes a value for `CSSFlagsFile`, by default **DITA2Go** creates a CSS file for flagged classes, places this file in your output directory, and causes the flags file to be referenced in the `<head>` section after your main CSS file.

To prevent **DITA2Go** from writing a CSS flags file:

```
[ConditionOptions]
; WriteFlagsFile = Yes (default, write in output directory) or
; No (do not write)
WriteFlagsFile = No
```

To prevent **DITA2Go** from including a reference to a CSS flags file:

```
[ConditionOptions]
; ReferenceFlagsFile = Yes (default, reference after main CSS file
; in output document head) or No (do not reference).
ReferenceFlagsFile = No
```

By default, the CSS file for flags remains a separate file; however, you can have **DITA2Go** append the CSS flags file to your main CSS file:

```
[ConditionOptions]
; AppendFlagsFile = No (default, keep separate from usual CSS) or
; Yes (append to regular CSS file if that file is being written)
AppendFlagsFile = Yes
```

*JavaHelp CSS  
breaks flagging*

If you are producing JavaHelp, be aware that JavaHelp does not recognize multiple classes, but instead uses the first class encountered in the CSS file (regardless of the order in the `class` attribute itself) and ignores the rest. This breaks flagging for everything except image (where **DITA2Go** inserts the image in the HTML output).

See also:

§9.2 [Defining conditional actions](#) on page 162

## 9.4 Configuring conditional flags

If you have set conditional-processing flags in configuration-file section `[ConditionalFlagging]`, for those flags to have an effect in output you must either define their properties or assign them appropriate attributes.

In this section:

§9.4.1 [Specifying text properties for flags](#) on page 166

§9.4.2 [Providing images and alt text for startflag and endflag](#) on page 168

§9.4.3 [Highlighting conflicting flag properties](#) on page 168

### 9.4.1 Specifying text properties for flags

To specify the properties of a conditional flag:

```
[ConditionalFlags]
flagname = property1 property2=value property3 ...
```

The *property* and *property=value* assignments are space delimited.

Properties you can assign to *flagname* include:

**Table 9-1 Text properties for flags**

Type of property	Property name	Description
Colors	fgcolor	Foreground color
	bgcolor	Background color
	chcolor	Change-bar color
Typographics	underline	Normal text underline
	doubleline	Double underline
	numunderline	Numeric underline
	overline	Text overline
	strike	Strikethrough
	italics	Italic; can be combined with <b>bold</b>
	bold	Bold; can be combined with <i>italics</i>
Change bars*	change	Show change bars
	chcolor	Change-bar color
	chchars	Change-bar character(s); punctuation only
Images*	startflag	Image (or text) at start of flagged element
	endflag	Image (or text) at end of flagged element

\* Not implemented yet

**Colors** For colors, the property name is followed by an equals sign, then by the name of a color, or by the #RRGGBB hex value of the color. Color names you can specify include:

aqua	gray	navy	silver
black	green	olive	teal
blue	lime	purple	white
fuchsia	maroon	red	yellow

For example:

```
[ConditionalFlags]
whitefore = fgcolor=white
blueback = bgcolor=blue
greenback = bgcolor=green
```

A flag may have no more than one foreground color and one background color; if both are assigned, the colors must be different. If the colors are the same, a conflict occurs, and the conflicting properties are replaced by the properties of the `conflict` flag; see §9.4.3 [Highlighting conflicting flag properties](#) on page 168.

**Typographics** Line properties `underline`, `overline`, and `strike` can be combined for a single flag. However, `underline`, `doubleline` and `numunderline` cannot be combined. Properties `bold` and `italics` can be combined for a single flag; they can also be combined with a line property.

For example:

```
[ConditionalFlags]
deleted = italics strike fgcolor=red
```

**Change bars** *(Not implemented yet.)*

To show change bars, assign property change to a flag. When you assign change, you can also assign the following properties to the change-bar flag:

- a color, with property `chcolor`
- one or more punctuation characters, with property `chchars`; these are displayed in the margin, for output types that do not support change bars *per se*.

Property `chcolor` may be any of the colors listed under [Colors](#), or the #RRGGBB hex value of the color. Property `chchars` may consist of punctuation characters only.

For example:

```
[ConditionalFlags]
altered = change chcolor=teal chchars=!*~
```

**Images** You can use a conditional flag to precede and follow an element with images or with text. This can be another way to substitute for change bars in output types such as HTML that do not support change bars. The idea would be to start the revised element with an icon something like >> and end it with <<; or to specify surrounding text such as *Start of change* and *End of change*, perhaps in a contrasting color.

For example:

```
[ConditionalFlags]
altered = startflag endflag fgcolor=red bold
```

When you assign `startflag` and `endflag`, you must also provide links to the images, alternate text to display, or both; see §9.4.2 [Providing images and alt text for startflag and endflag](#) on page 168.

## 9.4.2 Providing images and alt text for startflag and endflag

For DITA2Go to make startflag and endflag (see §9.4.1 [Specifying text properties for flags](#) on page 166) display something in output, you must specify the image, the text, or both to display:

[Starting image](#)

[Starting text](#)

[Ending image](#)

[Ending text](#)

**Starting image** To specify an image for a conditional flag assigned property startflag:

```
[CondStartFlagImages]
; flag name = href for an image that marks the start of the flagged
; area, usually a relative path or a URL for HTML
flagname = reference/to/image.gif
```

This setting takes effect only if the startflag property is assigned to *flagname* in [ConditionalFlags]; see §9.4.1 [Specifying text properties for flags](#) on page 166.

**Starting text** To specify alternate text for a conditional flag assigned property startflag:

```
[CondStartFlagAltText]
; flag name = alt text for the start flag image
flagname = Start of change
```

If no image is assigned to *flagname* in [CondStartFlagImages], the text assigned here is displayed instead. This setting takes effect only if the startflag property is assigned to *flagname* in [ConditionalFlags]; see §9.4.1 [Specifying text properties for flags](#) on page 166.

**Ending image** To specify an image for a conditional flag assigned property endflag:

```
[CondEndFlagImages]
; flag name = href for an image that marks the end of the flagged
; area, usually a relative path or a URL for HTML
flagname = reference/to/image.gif
```

This setting takes effect only if the endflag property is assigned to *flagname* in [ConditionalFlags]; see §9.4.1 [Specifying text properties for flags](#) on page 166.

**Ending text** To specify alternate text for a conditional flag assigned property endflag:

```
[CondEndFlagAltText]
; flag name = alt text for the end flag image
flagname = End of change
```

If no image is assigned to *flagname* in [CondEndFlagImages], the text assigned here is displayed instead. This setting takes effect only if the endflag property is assigned to *flagname* in [ConditionalFlags]; see §9.4.1 [Specifying text properties for flags](#) on page 166.

## 9.4.3 Highlighting conflicting flag properties

Flag name `conflict` is reserved for cases where different flags are set for the same element, and their properties cannot be merged without loss. For example, a conflict occurs when two such flags are both assigned the same color property but with different values; or when the value of `fgcolor` is the same as the value of `bgcolor`, either for the same flag or for different flags that apply to the same element.

Probably you would want to assign noticeable properties to the conflict flag, so the problem stands out. For example:

```
[ConditionalFlags]
conflict= bgcolor=white fgcolor=#ff00ff overline underline
```

## 9.5 Assigning attributes with conditional flags

To assign an attribute to conditionally flagged elements:

```
[ConditionAttributes]
; flag name = attribute name=value pair for output, usually class
flagname = attrname="value"
```

For example:

```
[ConditionAttributes]
blueback= class="bluebg"
```

Typically this setting is used to assign a CSS class name, which is appended to any other class attribute so that the flag properties can be defined in CSS. This setting has no effect in RTF output.

For HTML output only, the names of any flags to which you have *not* explicitly assigned a class attribute here become the class names themselves (`class="flagname"`). If an element has multiple unassigned flags, all are applied; for the HTML `class` attribute, that means all the class names appear as a space-delimited list in the attribute. Unassigned flags are ignored for XML output.

## 9.6 Scoping and filtering within maps

With DITA version 1.3 and specializations you should be able to apply conditions to parts of maps. You can achieve some of this effect in DITA version 1.1 or 1.2 with PIs that constrain the scope of ditaval conditions; and as a bonus, you can reuse topics in multiple places in the same document.

**Note:** Ordinarily, DITA PIs should be used only for filterable events; see §38.1.3 [Deciding when to use PI markers](#) on page 718. Including or omitting the PIs described in this section would affect content, not just presentation, and therefore should be considered a workaround. To preserve interoperability, use them only until the DITA standard includes an architectural feature to address the need for filtering within maps.

*In this section:*

§9.6.1 [Understanding the advantages of filtering in maps](#) on page 169

§9.6.2 [Designating map sections as named branches](#) on page 170

§9.6.3 [Reusing the same topics with different conditions](#) on page 170

§9.6.4 [Directing a cross reference to the correct branch](#) on page 171

§9.6.5 [Directing a content reference to the correct branch](#) on page 171

§9.6.6 [Limiting the scope of keydefs by branch](#) on page 172

§9.6.7 [Directing a key reference to the correct branch](#) on page 173

### 9.6.1 Understanding the advantages of filtering in maps

Filtering on `<topicref>` elements in a map instead of on attributes in a topic can help keep topics reusable, by not cluttering them with attributes that must work correctly in all maps in which those topics will ever be referenced. Filtering on attributes in topics can be a problem for another reason: if a `<topicref>` pulls in a topic, and you then filter out the content of that topic, you are left with something that might not be valid.

*Same content for multiple outputs in same map*

Suppose a topic includes information common to more than one product (for example), with no variations in content by product. And suppose two or more products are included in the map that references this topic. You must make sure the cross references from other topics reference the correct instance of the topic for each product. You cannot just reference one instance, because then you lose parent/child and previous/next relationships that include the common topic.

*Varying content for multiple outputs in same map*

Suppose the topic information varies to some degree for each product. That means the topic must be filtered by a different ditaval file for each instance. **DITA2Go** accomplishes this by adding a ditaval attribute to a PI that identifies the variant.

## 9.6.2 Designating map sections as named branches

To specify that part of a bookmap or ditamap should be treated differentially, you can insert a special PI marker to give that map section a “branch” name:

```
<?dtall branch="branchname" ?>
```

See §38 [Working with processing instructions](#) on page 717.

You do not need to specialize, or do anything else with the DITA code itself. When you place a branch PI marker in a map (not in topics), doing so marks everything in the map that comes after the PI marker, at the same level or below, as belonging to that branch.

You can override a branch PI with another branch PI marker, at the same level or below, following the same scoping rules (inheritance). **DITA2Go** manages branch PIs on a stack, so restore works as expected.

## 9.6.3 Reusing the same topics with different conditions

Suppose you want to use the same map in two places in your output, with different conditions applied. If (for example) you insert the following branch PI marker in a map:

```
<?dtall branch="productA" ditaval="producta.ditaval" ?>
```

then all topics referenced in branch `productA`, and no others, acquire the conditions specified in `producta.ditaval`. But you can also insert:

```
<?dtall branch="productB" ditaval="productb.ditaval" ?>
```

in the same map, with the very same files in scope, and with two `<topicref>` elements get the two outputs in one document. In each case, you would insert the branch PI marker just before the `<topicref>` you want it to influence.

For example, you might set up a map like this:

```
<map>
  <topicref href="common_intro.dita" />
  <topichead navtitle="Product A">
    <?dtall branch="productA" ditaval="producta.ditaval" ?>
    <topicref href="product.ditamap" format="ditamap" />
  </topichead>
  <topichead navtitle="Product B">
    <?dtall branch="productB" ditaval="productb.ditaval" ?>
    <topicref href="product.ditamap" format="ditamap" />
  </topichead>
</map>
```

The very same map is used for both branches. So you get two copies, each filtered by its own ditaval. You have complete flexibility to reuse a topic any number of times in the same project, with different conditions for each use. You can apply a different ditaval to each branch, or the same ditaval to disjoint areas of the map.



This method of accommodating duplicated topics avoids the workaround of reverse conref elements, which is the usual (and ugly) answer: create empty topics for each place of use, and conref the real information from a library of topics, as opposed to just referencing the topic you want to use in the map.

### 9.6.4 Directing a cross reference to the correct branch

Branch PI markers in maps allow you to specify inclusion of the same topic in more than one place in the output (see §9.6.3 [Reusing the same topics with different conditions](#) on page 170). How do you specify where a cross reference to a duplicated topic should go? **DITA2Go** applies two rules, in the following order:

1. If a cross reference in a named map branch targets another topic in the same branch, the cross reference goes to the instance in its own branch.
2. If a cross reference is to a topic in a different named branch, you can specify the target with a cross-reference branch PI marker placed just before the cross reference:

```
<?dtall xrefbranch="branchname" ?>
```

A cross-reference branch PI applies only to the very next cross reference.

If neither of these rules applies, the cross reference goes to the first instance of the topic.

Suppose in `common_intro.dita` you want to reference an introduction that occurs in each branch. You would insert a cross-reference branch PI in the referencing topic, just before each cross reference. For example:

```
<p>The following products are described here:
<ul>
  <?dtall xrefbranch="productA" ?>
  <li><xref href="product_intro.dita" /></li>
  <?dtall xrefbranch="productB" ?>
  <li><xref href="product_intro.dita" /></li>
</ul>
</p>
```

The references are both to the same file; the PIs sort out which instance of the file to target. So if the file referenced has conditions applied to `<ph>` elements in its title to yield different resulting titles, you get the correct title for each in the list items.

### 9.6.5 Directing a content reference to the correct branch

When you include the same topic in more than one place in the output, to pull in content from that topic via conref, you must be able to designate the correct instance; you do this by specifying the map branch. To resolve a conref, **DITA2Go** applies the same rules as for cross references (see §9.6.4 [Directing a cross reference to the correct branch](#) on page 171):

1. If a conref in a named map branch targets another topic in the same branch, the content is pulled from the instance in its own branch.
2. If a conref is to a topic in a different named branch, you can specify the branch with a conref branch PI marker placed just before the element with the conref:

```
<?dtall conrefbranch="branchname" ?>
```

A conref branch PI applies only to the very next element that has a conref.

If neither of these rules applies, the conref pulls content from the first instance of the topic.

So, to reuse descriptions from the duplicated topics, you can set up the `<shortdesc>` elements in `product_intro.dita` as follows:

```

<concept id="prodintro">
  <title>Name of product with conditional ph
    elements as needed</title>
  <shortdesc><ph id="description">Text here, with
    conditional ph elements.</ph></shortdesc>
  ...
</concept>

```

Then:

```

<p>The following products are described here:
<ul>
  <?dtall xrefbranch="productA" conrefbranch="productA" ?>
  <li><xref href="product_intro.dita" />: <ph
    conref="product_intro.dita#prodintro/description"/></li>
  <?dtall xrefbranch="productB" conrefbranch="productB" ?>
  <li><xref href="product_intro.dita" />: <ph
    conref="product_intro.dita#prodintro/description"/></li>
</ul>
</p>

```

The two <li> elements are identical; the PIs direct each one to its own branch, so they pull in different content based on the conditions applied to those branches.

### 9.6.6 Limiting the scope of keydefs by branch

With named branches and branch PIs you can limit the scope of keydefs, and also determine whether global keydefs should be used as a backup to branch keydefs.

To give preference to keydefs in a named branch when resolving keyrefs in that branch:

```

[KeyOptions]
; UseBranchKeydefs = No (default, standard DITA 1.2 single key space
; for whole map), or Yes (within a branch, search for keydefs in the
; same branch before searching the global key space)
UseBranchKeydefs = Yes

```

When UseBranchKeydefs=Yes, if a named branch (always in a map) contains keydefs, if possible **DITA2Go** uses those keydefs to resolve keyrefs in that branch. If it is not possible to do so for a given keyref, or when UseBranchKeydefs=No, **DITA2Go** uses the normal DITA 1.2 keydef resolution precedence: top down, breadth first from the root map, first instance of the sought key prevails.

To restrict the use of branch keydefs to their named branch:

```

[KeyOptions]
; KeydefsOnlyWithinBranch = No {always retain keydefs in branches in
; the global key space}, or Yes (exclude branch keydefs from the
; global key space when UseBranchKeydefs=Yes).
KeydefsOnlyWithinBranch = Yes

```

When KeydefsOnlyWithinBranch=Yes, the keydefs in any named branch are used only in that branch. When KeydefsOnlyWithinBranch=No, all keydefs in named branches remain in the global key space; however, they are used preferentially within their named branch, provided UseBranchKeydefs=Yes.

For example:

```

<chapter id="ditaval1" navtitle="Ditaval Selection and Flagging">
  <?dtall branch="Original" ditaval="WindowsProps.ditaval" ?>
  <topicref href="DITAProps.ditamap" format="ditamap" />
  <keydef keys="OS" ><topicmeta>
    <linktext>Windows</linktext>
  </topicmeta></keydef>
</chapter>

```

```

<chapter id="ditaval2"
navtitle="Ditaval Selection and Flagging Revisited">
<?dtall branch="Revisited" ditaval="LinuxProps.ditaval" ?>
<topicref href="DITAProps.ditamap" format="ditamap" />
<keydef keys="OS" ><topicmeta>
  <linktext>Linux</linktext>
</topicmeta></keydef>
</chapter>

```

When a topic within the referenced map contains:

```
<term keyref="OS" />
```

that term becomes “Windows” in the first chapter and “Linux” in the second chapter.

Suppose that after the two keydefs in the named branches, the map contains:

```

<keydef keys="OS" ><topicmeta>
  <linktext>Solaris</linktext>
</topicmeta></keydef>

```

When `KeydefsOnlyWithinBranch=Yes`, this code in a topic that is *not* in a named branch:

```
<term keyref="OS" />
```

yields “Solaris”. However, when `KeydefsOnlyWithinBranch=No`, the term becomes “Windows” (from the first keydef in the map).

## 9.6.7 Directing a key reference to the correct branch

To specify the branch to use for resolving the next keyref in a map or a topic, insert a **KeyrefBranch** PI:

```
<?dtall keyrefbranch="BranchName" ?>
```

A **KeyrefBranch** PI specifies the name of the branch to search to resolve the next keyref only, either in a map or in a topic. It works the same way as cross-reference branch PIs and conref branch PIs; see:

§9.6.4 [Directing a cross reference to the correct branch](#) on page 171

§9.6.5 [Directing a content reference to the correct branch](#) on page 171

Using the example in §9.6.6 [Limiting the scope of keydefs by branch](#) on page 172, if a topic contains:

```
<?dtall keyrefbranch="Revisited" ?><term keyref="OS" />
```

the term becomes “Linux” even if the keyref is in the Windows chapter.



# 10 Including content by reference

**DITA2Go** supports content reference push and pull, and inclusion of external text content. Topics include:

§10.1 [Pushing and pulling content by reference](#) on page 175

§10.2 [Referencing external code or text fragments](#) on page 176

## 10.1 Pushing and pulling content by reference

**DITA2Go** supports the conref push and pull actions included in DITA version 1.2, with a few restrictions and a couple of extensions.

*In this section:*

§10.1.1 [Referencing internal and external maps and topics](#) on page 175

§10.1.2 [Pushing content into an element](#) on page 176

§10.1.3 [Understanding problems with processing conkeyrefs](#) on page 176

### 10.1.1 Referencing internal and external maps and topics

Processing requirements limit what **DITA2Go** can do to support copying content from and to internal and external maps and topics. [Table 10-1](#) summarizes the restrictions.

**Table 10-1** DITA2Go support for push and pull conrefs

Conref from:	to:	Internal target		External target		Restrictions
		Push	Pull	Push	Pull	
map	map	No	No	No	Yes	Pull only, from external map only
map	topic	Yes	No	No	Yes	No internal pull; pushreplace only
topic	map	No	Yes	No	Yes	No push from topic to map
topic	topic	Yes	Yes	No	Yes	All five push variants

Nothing external to your project can be processed, except to pull from as a source. In other words, **DITA2Go** does *not* handle any of the following:

- push from a source external to your project
- push to a destination external to your project
- pull from an external destination.

*Conref from map to map*

Map-to-map conrefs can be pull only (no push), and only from a map external to the project. You can use map-to-map pull topicrefs to pull in topicrefs from another map, to provide a form of indirection that does not employ keyrefs, which are not included in DITA version 1.1. You can swap in different maps containing topicrefs with the same IDs but different attributes; and the main map will add topicrefs from the other maps at run time.

For example, if you have product-specific information to be included in many different places, you can have a topicref with a conref at each such place. However, you cannot use the same topic in the same map in two places, unless you use branches to distinguish them; see §9.6 [Scoping and filtering within maps](#) on page 169.

**Note:** Map to map conkeyrefs are not supported, because it is not possible to resolve keyrefs until maps are totally resolved, which cannot be done if any conkeyref is

present that can add a map; see §10.1.3 [Understanding problems with processing conkeyrefs](#) on page 176.

<i>Conref from map to topic</i>	A map cannot pull from a topic. By the time the topic is processed it is too late to affect map processing in any useful way, even from external topics. Map processing has to be finished (to resolve keyrefs) before any conrefs, and after that, changing the original elements cannot affect the already processed results. Only an internal map can push, and only pushreplace, not pushafter or pushbefore.
<i>Conref from topic to map</i>	The topic with the conref must be internal to the project, and can pull from either an internal or an external map. However, a topic cannot push to a map, because map processing is finished before topic processing begins.
<i>Conref from topic to topic</i>	Push and pull are supported between internal topics. Pull is also supported from external topics, but you cannot push to an external topic. Push includes all five variants, two of which (pushatstart and pushatend) are extensions not present in the DITA 1.2 specification; see §10.1.2 <a href="#">Pushing content into an element</a> on page 176.

## 10.1.2 Pushing content into an element

In addition to the three DITA 1.2 push conactions (pushreplace, pushbefore, and pushafter), **DITA2Go** also supports pushatstart and pushatend, which place the content inside the target location; that is, at the start or the end of the element content, respectively. The pushatstart and pushatend extensions put in only plain text extracted from the element with the conref. They are only topic-to-topic, not to/from or between maps.

The target must be an element that can contain text, but is not otherwise limited. Only the text content of the element with the conref is pushed. You can push text to the same element as many times as you want; the pushes are performed in document order (of the elements with the conrefs), and result in preserving that order both at start and at end. That is, if you push A, then B, to start and to end, you get AB ... AB (not BA ... AB).

## 10.1.3 Understanding problems with processing conkeyrefs

**DITA2Go** processes conkeyrefs at the same time as keyrefs, after reading all the original maps and before processing any map conrefs. Adding maps by conref after that has the potential to change the keydefs. For example, if the top level map conrefs a topicref in an external map that contains keydefs, those keydefs would theoretically be higher priority than any referenced in lower-level maps. But the keyrefs in all the existing maps have already been resolved, so there is a contradiction. **DITA2Go** handles this problem by disregarding keydefs found in conref'd topicrefs.

If the keydefs are in the original set of maps, they will be used to resolve any conkeyrefs in those maps, and that will happen before any new maps are conref'd. For that case, conkeyref is identical to conref. But if a conref'd map element itself has a conkeyref, **DITA2Go** still uses the original keydefs to resolve it. If the new element has a key not present in the original set, but perhaps present in itself, that new keydef will not be used. So in that unusual case a conkeyref may or may not work, but a conref will work.

## 10.2 Referencing external code or text fragments

Proposed Internet standard RFC 5147 provides a fragment identifier scheme for plain text documents, which allows reference to both ranges of characters and ranges of lines within a plain text document:

<http://tools.ietf.org/html/rfc5147>

**DITA2Go** supports the proposed `href` attribute syntax, and also provides PI markers to achieve the same effect. We recommend using the fragment identifiers described in RFC 5147 rather than PI markers, where it makes sense to do so.

*In this section:*

§10.2.1 [Including external code snippets with PI markers](#) on page 177

§10.2.2 [Including external code snippets with fragment identifiers](#) on page 177

## 10.2.1 Including external code snippets with PI markers

When you use the DITA 1.2 `<coderef>` element to include code from an external file, by default you get the content of the entire referenced file. However, you can use **DITA2Go** external-code PI markers to select only a portion of the referenced code, by character range or by line number. For example:

```
<?dtall ExtCodeStartLine="4" ExtCodeEndLine="15" ?>
```

Placed in your document, this PI marker would include only lines 4 through 15 of the file referenced in the next `<coderef>`. To start at the beginning of the referenced file, omit `ExtCodeStartLine`; to go to the end, omit `ExtCodeEndLine`. You can specify starting and ending characters in the file, instead of lines; for example:

```
<?dtall ExtCodeStartChar="34" ExtCodeEndChar="320" ?>
```

You could include a check on the length of the file, and specify its encoding, also:

```
<?dtall ExtCodeFileLen="438" ExtCodeFileEnc="UTF-8" ?>
```

An external-code PI marker applies only to the next `<coderef>` instance, and is self canceling following the `<coderef>`. Instead of `dtall`, you can use `dthtml` to affect only HTML/XHTML/XML outputs, or `dttrtf` to affect only Word and WinHelp output.

See §38 [Working with processing instructions](#) on page 717.

## 10.2.2 Including external code snippets with fragment identifiers

You can use RFC 5147 fragment identifier syntax in the `href` attribute value to select portions of code or text from an external file. The following two examples produce the same result:

*With a processing instruction*

```
<?dtall ExtCodeStartLine="4" ExtCodeEndLine="15"
  ExtCodeFileLen="438" ExtCodeFileEnc="UTF-8" ?>
<codeblock><coderef href="fdkfunc.c" /></codeblock>
```

*With a fragment identifier*

```
<codeblock><coderef
  href="fdkfunc.c#line=3,15;length=438,UTF-8" /></codeblock>
```

The fragment identifier is not required to have `#topicID/elemID` in `<coderef>` because the target file is not necessarily DITA, so the latter usage does not conflict with the DITA specification.

In RFC 5147, the starting line number is actually the line number *before* the first line you want, because it is zero-based; to start with the first line, you would use `#line=,15`. By contrast, **DITA2Go** PI markers remain one-based, just as all text editors do.

RFC 5147 does not say what to do if both line and character offsets are present. **DITA2Go** starts with the starting line, then counts to the starting character from that point, and continues until the ending character (relative to the same point), or the ending line, whichever is encountered first. This allows you to select a part of one or more lines easily, without having to count characters (which are code points, not bytes) to get to the starting line.

The file length (in bytes) provides a way to check whether the file has changed since you specified the offsets. This is the same check Norm Walsh makes in Calabash for the XInclude version. If the file size is provided and is not the same, **DITA2Go** logs an error and does not include any file content, per RFC 5147.



# 11 Defining element sets and properties

---

This section shows how to aggregate elements into sets for assigning properties, and how to characterize the properties of specialized DITA elements. Topics include:

§11.1 [Defining sets of elements](#) on page 179

§11.2 [Specifying properties of element types](#) on page 179

See also:

§B [Element type default properties](#) on page 825

## 11.1 Defining sets of elements

To reduce the number of variants of essentially the same hierarchy, when you define an element path (see §6.4.1 [Understanding element paths](#) on page 92), you might want to treat several elements as equivalents; for example, topic, concept, task, and reference. You can define a single *element set* that includes them all, and reference that element set in the element path.

To define sets of elements:

```
[ElementSets]
; Name for set = list of elements and element sets, separated by
; spaces.
$setname = element1 element2 $otherset
```

Give each set an alphanumeric name prefixed with a \$ (dollar sign). List the names of members of the set to the right of the equals sign, separated by spaces. For example:

```
[ElementSets]
$topic = topic concept task reference glossentry
$body = body conbody taskbody refbody glossdef
$sect = section example refsyn
$map = map bookmap
$tref = topicref topichead chapter frontmatter backmatter
```

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.

Member sets must be defined in [ElementSets] above any other sets that include them. Keep the definition of each set all on one line in your configuration file.

## 11.2 Specifying properties of element types

**DITA2Go** defines more than 50 properties that can be assigned to elements to characterize the roles those elements play in a conversion project. These properties go beyond the attributes specified in a DITA DTD.

*In this section:*

§11.2.1 [Understanding when to assign element type properties](#) on page 180

§11.2.2 [Understanding what properties are available](#) on page 180

§11.2.3 [Assigning properties to element types](#) on page 183

§11.2.4 [Adding the full class attribute to an element](#) on page 183

## 11.2.1 Understanding when to assign element type properties

All elements in the base DITA DTDs have preassigned **DITA2Go** properties, listed in §B [Element type default properties](#) on page 825. You need to add or modify property assignments only to accomplish one of the following:

- change a default, if the default properties are not in accord with the way you are using an element
- specify properties of specialized elements.

For the former, as an example you might want to use the value of a particular element as a variable; see §12.1 [Understanding how DITA2Go user variables work](#) on page 185. If that element does not already have property `Var` by default, you would need to add `Var` to the list of properties for the element.

For the latter, if you do not specify properties, a specialized element inherits the properties of the element from which it was derived.

If your DITA document includes specialized elements with presentational characteristics different from those of the base elements from which they are derived, you might want to assign appropriate properties to the specialized elements.

On the other hand, if you convert specialized documents *without* defining element type properties, and you like the results, you need not specify these properties for your specialized elements. That will happen if you have specialized from base elements that have the *same* presentational characteristics as your specialized elements. This is the DITA process for generalization, which is what makes specialization so useful: tools that do not understand a specialized element are required to treat it as the highest base class they do understand. That handles most cases.

## 11.2.2 Understanding what properties are available

[Table 11-1](#) shows the properties you can assign to the full class attribute of an element. The list of elements for which a property is assigned by default is representative, not exhaustive; for the complete list of default property assignments, see §B [Element type default properties](#) on page 825.

Properties are grouped as follows in [Table 11-1](#):

Basic	Inline, Pre, Text, Typo, Num, Data, Pernicious, NoLevel, Draft
Role	Root, Map, Topic, Meta, Var, Title, Section, Note, Task, Reference, Glossary, BookTitle, Navtitle, Trademark
Image or object	Fig, Image, Alt, Object, Param
Index related	Index, IxStart, IxSee, IxAlso, IxSort
Link or reference	Ref, Key, XRSource, Link, Rel, Abbrev
List related	List, ListItem, DL*, Sub, CascadeSet, CascadeItem, Numbered, Nonum, PL*
Grouping	Group, Sequence
Descriptive	Footnote, Desc, SDesc, Abstr
Table related	Table, TabStart, TabCol, TabRow, TabCell, TabHead, TabBody

**Table 11-1 Element type properties**

Group	Property	Description
Basic	Inline	Does not start a new paragraph, starts an inline (character) format, unless the element is also assigned property <code>Typo</code> . Default for <code>ph</code> , <code>img</code> , <code>xref</code> , <code>fn</code> , <code>tt</code> .
	Pre	May be block or Inline, always <code>Text</code> ; retains all existing whitespace. Default for <code>pre</code> , <code>lines</code> , <code>codeblock</code> .
	Text	Can have text; true of all Inline and of some block. An empty block element that is not also <code>Text</code> is deleted; otherwise it becomes an empty paragraph. Default for <code>p</code> , <code>lq</code> , and many others.
	Typo	Typographic property only; always Inline, but does not start an inline format; treated like an override instead. Default for <code>b</code> , <code>i</code> , <code>u</code> , <code>sub</code> , <code>sup</code> .
	Num	Always given a number. Default for <code>ol</code> , <code>fn</code> .
	Data	Suppressed for output, except for any contained text; normally used for metadata. Default for <code>data</code> , <code>data-about</code> .
	Pernicious	Can contain both text ( <code>#PCDATA</code> ) and block (as opposed to inline) elements, resulting in “pernicious mixed content”. Default for <code>li</code> and <code>entry</code> .
	NoLevel	Does not add to the hierarchy of levels for elements such as headings. Default for <code>sectiondiv</code> , <code>bodydiv</code> , <code>conbodydiv</code> , and <code>refbodydiv</code> .
	Draft	Used only for <code>&lt;draft-comment&gt;</code> .
Role	Root	Root element for DITA map or topic. Default for <code>topic</code> , <code>concept</code> , <code>task</code> , <code>reference</code> , <code>glossentry</code> , <code>map</code> , <code>bookmap</code> .
	Map	Specific to maps. Default for <code>map</code> and <code>bookmap</code> .
	Topic	Map element containing topic information. Default for <code>topichead</code> , <code>topicref</code> , <code>topicmeta</code> .
	Meta	Contains metadata. Default for <code>topicmeta</code> .
	Var	Sets a variable. Default for many bookmap text metadata elements: <code>author</code> , <code>booknumber</code> , <code>booktitlealt</code> , <code>brand</code> , <code>category</code> , <code>component</code> , <code>copyrholder</code> , <code>copyright</code> , <code>copyryear</code> , <code>featnum</code> , <code>isbn</code> , <code>mainbooktitle</code> , <code>platform</code> , <code>prodname</code> , <code>prognum</code> , <code>publisher</code> , <code>series</code> , <code>source</code> ; see §12 <a href="#">Creating and deploying user variables</a> on page 185
	Title	Used for title. Default for <code>title</code> .
	Section	Used to group paragraphs. Default for <code>section</code> .
	Note	Element with <code>type</code> attribute, such as <code>note</code> . Default for <code>note</code> .
	Task	For <code>&lt;task&gt;</code> specific elements; also used for <code>&lt;example&gt;</code> .
	Ref	For <code>&lt;reference&gt;</code> specific elements.
	Glossary	Used for <code>&lt;glossentry&gt;</code> specific elements, <code>&lt;term&gt;</code> , and <code>&lt;abbreviated-form&gt;</code> .
	BookTitle	Used for <code>&lt;mainbooktitle&gt;</code> and <code>&lt;booktitlealt&gt;</code> .
	NavTitle	Default for <code>&lt;navtitle&gt;</code> .
	Trademark	Default for <code>&lt;tm&gt;</code> .
Image or object	Fig	Contains a complete figure. Default for <code>fig</code> .
	Image	Contains image and <code>alt</code> text. Default for <code>image</code> .
	Alt	Contains <code>alt</code> text for image. Default for <code>alt</code> .
	Object	Contains a media object. Default for <code>object</code> .
	Param	Parameter element for an object. Default for <code>param</code> .

**Table 11-1 Element type properties (continued)**

Group	Property	Description
Index related	Index	Used for indexing. Default for <code>indexterm</code> and <code>index*</code> .
	IxStart	Starts an index entry. Default for <code>indexterm</code> .
	IxSee	Index see reference. Default for <code>index-see</code> .
	IxAlso	Index see-a/so reference. Default for <code>index-see-also</code> .
	IxSort	Index sort string. Default for <code>index-sort-as</code> .
Link or reference	Ref	Used for cross-referencing. Default for <code>xref</code> ., <code>topicref</code> , <code>booklist</code> , among many others.
	Key	Use for DITA 1.2 indirection. Default for <code>keydef</code> .
	XRSource	Contains a number or content that may be used in an <code>xref</code> . Default for <code>title</code> , <code>li</code> , <code>fn</code> .
	Link	Related Topics link. Default for <code>link</code> .
	Rel	Related information. Default for <code>related-links</code> , <code>reltable</code> .
	Abbrev	Used for keyrefs to abbreviations. Default for <code>abbreviated-form</code> .
Grouping	Group	Wraps a group. Default for <code>topicgroup</code> .
	Sequence	Wraps an ordered set. Default for <code>linkinfo</code> , <code>linklist</code> .
List related	List	Contains a list; default for <code>sl</code> , <code>ol</code> , <code>ul</code> .
	ListItem	Used within a list; default for <code>li</code> , <code>sl.i</code> .
	DList	Definition list start; default for <code>dl</code> .
	DLEntry	Entry in definition list; default for <code>dlentry</code> .
	DLTerm	Definition list term; default for <code>dthd</code> , <code>dt</code> .
	DLDef	Definition list definition; default for <code>ddhd</code> , <code>dd</code> .
	Sub	Distinguish substeps from steps; default for <code>substeps</code> .
	CascadeSet	Default for <code>menucascade</code> .
	CascadeItem	Default for <code>uicontrol</code> .
	NoNumber	Default for <code>stepsection</code> .
	PList	Used for a parameter list.
	PLmEntry	Used for a parameter list component.
	PLTerm	Used for a parameter list component.
	PLDef	Used for a parameter list component.
Descriptive	Footnote	Contains a footnote; default for <code>fn</code> .
	Desc	Description used in object, etc.; default for <code>desc</code> .
	SDesc	Short description in <code>topicref</code> or <code>topic</code> ; default for <code>shortdesc</code> .
	Abstr	Abstract in <code>topic</code> ; default for <code>abstract</code> .
Table related	Table	Starts a table or contains table-wide information. Default for <code>table</code> , <code>simpletable</code> , <code>parml</code> , <code>dl</code> , <code>colspec</code> , <code>tgroup</code> , <code>thead</code> , <code>tbody</code> .
	TabStart	Starts a table. Default for <code>dl</code> , <code>parml</code> , <code>simpletable</code> , <code>table</code> .
	TabCol	Contains column properties. Default for <code>colspec</code> .
	TabRow	Contains a row. Default for <code>row</code> , <code>sthead</code> , <code>strow</code> , <code>plentry</code> , <code>dlhead</code> , <code>dlentry</code> .
	TabCell	Table cell. Default for <code>entry</code> , <code>stentry</code> , <code>pt</code> , <code>pd</code> , <code>dthd</code> , <code>ddhd</code> , <code>dt</code> , <code>dd</code> .
	TabHead	Occurs in table head. Default for <code>thead</code> , <code>sthead</code> , <code>dlhead</code> .
	TabBody	Occurs in table body. Default for <code>tbody</code> , <code>strow</code> , <code>dlentry</code> , <code>plentry</code> .

### 11.2.3 Assigning properties to element types

You must assign element type properties to the full class attribute of the element in question, because the same element name can be reused in another context; for example, both as a topic element and as a topicref in a map. You can set a maximum of 11 properties for any one element; **DITA2Go** ignores any additional properties.

To specify properties for element types:

```
[ElementTypes]
; full class attribute = properties to use for element
class/attribute of/element = property1 property2 ...
```

For example:

```
[ElementTypes]
topic/strow task/chrow = Task TabRow TabBody
map/topicref bookmap/colophon = Map Ref Topic
topic/component = Text Var
topic/copyryear =
```

These examples show a few of the default property assignments listed in §B [Element type default properties](#) on page 825.

An empty assignment means the element is block only (not inline). If an element does not have the `Inline` property, it is treated as block, and is assumed to start a new paragraph, or to contain paragraphs or higher aggregates. If an element has the `Text` property, it starts a block (paragraph) format. This is the default for all elements that do not have the `Inline` property.

### 11.2.4 Adding the full class attribute to an element

You might encounter a deficiency in the DTDs that in some cases prevents them from adding the `@class` to an element. **DITA2Go** adds the `@class` if possible, and otherwise processes such an element as a `<data>` element, and writes the following error message to the log file for your project:

```
No class attribute for element name
```

To correct this deficiency, for example for `<abbreviated-form>`:

```
[ElementClasses]
; Element name = "+ " followed by the full class attribute
; followed by a space.
abbreviated-form = + topic/term abbrev-d/abbreviated-form
```

**Note:** The trailing space is required, as is the space following the “+”.

If the DTD does in fact supply `@class` for an element, as it should, a setting in `[ElementClasses]` for that element will *not* override the DTD.



# 12 Creating and deploying user variables

**DITA2Go** can capture DITA element content, and in some cases attribute values, in named **DITA2Go** user variables for use in macros. This section describes the configuration settings required to create and deploy such variables. Topics include:

§12.1 [Understanding how DITA2Go user variables work](#) on page 185

§12.2 [Assigning variable names to element paths](#) on page 186

§12.3 [Including user variables in DITA2Go macros](#) on page 186

§12.4 [Deploying user variables in template macros](#) on page 187

## 12.1 Understanding how DITA2Go user variables work

**DITA2Go** provides the following ways to capture element content in variables that you can subsequently use in macros:

- Assign property `Var` to an element; see §11.2 [Specifying properties of element types](#) on page 179. The variable has the same name as the element.
- Assign the name of a variable to an element path; see §12.2 [Assigning variable names to element paths](#) on page 186.

Either way, what you get is a **DITA2Go** *user variable*. If an element is neither listed with a path in `[VariableMaps]` nor assigned property `Var` in `[ElementTypes]`, no user variable is created for it.

If you specify that an element is to function as a **DITA2Go** user variable, either by listing an element path in `[VariableMaps]` or by setting `Var` for the element in `[ElementTypes]` (explicitly or by default), **DITA2Go** stores the first instance as a regular user variable. If the same element or path occurs again in your document, the variable begins a list, and every subsequent instance is added to that list. A multiple-instance user variable behaves like a read-only list variable; see §37.4 [Using multiple-value list variables](#) on page 695.

*Accessing single-instance user variables*

You can access the content of a single-instance user variable in a macro as `<$$varname>`, provided you have not already defined a **DITA2Go** *macro* variable of the same name, either in `[MacroVariables]` (see §37.3 [Using macro variables](#) on page 687) or by assigning a value to that name in a macro. Elements assigned property `Var` by default (listed in [Table 11-1](#) on page 181) appear just once in each DITA document, in a bookmap. For these metadata items, only one value is captured, and you do not need to use an index number to access the value. For example, to use the content of `mainbooktitle` in a macro, you would reference `<$$mainbooktitle>`; see §37.3 [Using macro variables](#) on page 687.

*Accessing multiple-instance user variables*

You can access any instance of a multiple-instance user variable (after the first instance) as `<$$varname[index]>`, where *index* is a number from 1 to *N*. You can access the value of *N* with `<$$varname[]>`. Each value is referenced by specifying the name and the index-number-minus-1 of its occurrence. For example, if you (perhaps foolishly) assign property `Var` to element `title`, the content of the third instance of `title` in your document will be available as `<$$title[2]>` (because the first instance is `<$$title>` and the second is `<$$title[1]>`). The 568th instance will be `<$$title[567]>`. If you reference just `<$$title>`, you get the first instance.

*User variables are read-only*

You cannot alter the content of a user variable derived from a DITA element; these variables are read-only. However, you can override the content of a user variable by

assigning a value to a macro variable of the same name; thereafter, the user variable is no longer accessible.

## 12.2 Assigning variable names to element paths

You can assign a name to an element path to capture the element content in a **DITA2Go** user variable of that name.

To map an element path to a user variable:

```
[VariableMaps]
; element path = name to use for variable with element content,
; or None (variables are ignored, except booktitle and booktitlealt)
; default is the element name itself
elementpath = varname
```

For example:

```
[VariableMaps]
mainbooktitle/* = BookTitle
```

The user variable is then available for use in **DITA2Go** macros. This assignment applies to any element, regardless of the properties set for the element in `[ElementTypes]`. How many variables you get depends on how many instances of the element path occur in your document. For example, if you set:

```
[VariableMaps]
title/* = MainTitle
```

you will get a variable `$$MainTitle[index]` for every use of `title` in your document. If you change the setting to:

```
[VariableMaps]
title/map/1 = MainTitle
```

you will get just one variable, `$$MainTitle`. How specific you make the path determines how many matches you get.

When `elementpath=None` (or `No`), user variables are not created for that particular element path.

The **DITA2Go** distribution includes the following setting in system configuration file `%OMSYSHOME%\d2g\configs\d2g_config.ini`:

```
[VariableMaps]
* = None
```

If you omit all settings in `[VariableMaps]`, this is what you get. In this case the only user variables **DITA2Go** recognizes are for `booktitle` and `booktitlealt`, which are handled differently.

## 12.3 Including user variables in DITA2Go macros

Suppose you want to produce a list of titles in your document. If you want all titles, you could assign property `Var` to `title` (see §11.2 [Specifying properties of element types](#) on page 179). If you want only chapter titles, you could assign a user variable name to an element path that begins with `title`:

```
[VariableMaps]
title/$topic/1 = title
```

You could create a macro that iterates through the resulting `<$$title[n]>` multiple-instance user variable:



```
[WriteTitles]
<$$title><$_repeat ($$title[])><br>
<$$title[$$_wcount]><$endrepeat>\
```

and invoke the macro with code assigned to another macro, or to a configuration setting:

```
<p class="titles"><$WriteTitles></p>
```

to produce a list of titles:

```
<p class="titles">Test Suite Design<br>
Purpose<br>
Suite Components<br>
Expected Results<br>
Test Suite Operation<br>
Test Suite Coverage<br>
Contributors<br>
. . .
```

See §37.1 [Defining and invoking macros](#) on page 679.

## 12.4 Deploying user variables in template macros

One of the advantages of **DITA2Go** user variables is that they provide a way to produce additional output that is separate from the main document output stream, and that includes metadata. This allows you to do things such as construct title pages and generate lists. To produce such output, you deploy user variables in a template macro. A *template macro* is a macro whose name is assigned to an external file name. A *macro template* associates a template macro with a file name.

To assign a template macro to a file:

```
[MacroTemplates]
; filename.ext for template output doc = template macro name
filename.ext = macroname
; or:
filename.ext = path/to/macro/file
```

The name of the file to which you assign the template macro must include an extension, and may include a path. The default location for the file is the output directory.

The template macro name can refer to any of the following:

- a macro defined in your project configuration file
- a macro defined in a macro library file referenced by your configuration file (see §37.2.4 [Including macro definitions in your own macro library](#) on page 685)
- a file that contains only the template macro.

This example assigns a template macro named `$BookTitlePageTemplate` to an output file named `booktitlepage.htm` located in the output directory:

```
[MacroTemplates]
booktitlepage.htm = BookTitlePageTemplate
```

Variable `maptitle` is assigned to the map title for the document:

```
[VariableMaps]
title/map/1 = maptitle
```

The content of variable `author` (element `author` is automatically a variable; see property Var in [Table 11-1](#) on page 181) is replaced with a new value:

```
[MacroVariables]
author = Omni Systems
```

The template macro is defined in the configuration file:

```
[BookTitlePageTemplate]
<html><head>
<title><$$maptitle></title>
</head><body>
<h1><$$maptitle></h1>
<p class="author">By <$$author></p>
</body></html>
```

When `<$$BookTitlePageTemplate>` is invoked, **DITA2Go** writes the following to file `booktitlepage.htm` in the output directory:

```
<html><head>
<title>DITA Test Suite</title>
</head><body>
<h1>DITA Test Suite</h1>
<p class="author">By Omni Systems</p>
</body></html>
```

# 13 Processing related and associative links

This section shows what choices you have for getting the content of DITA related links and other links into your output. Topics include:

- §13.1 [Understanding how DITA2Go treats reltables](#) on page 189
- §13.2 [Generating and including related links](#) on page 189
- §13.3 [Appending links to topics](#) on page 190
- §13.4 [Including descriptions with related links](#) on page 191
- §13.5 [Generating associative links for Help output](#) on page 192
- §13.6 [Formatting links in output](#) on page 192
- §13.7 [Changing link path for peer related links](#) on page 196

## 13.1 Understanding how DITA2Go treats reltables

**DITA2Go** assumes a reltable describes all relationships you might ever want for topics in the map; or potentially in the map, perhaps under different conditions.

This “permissive” view of reltables allows them to provide a “superset” of relationships, from which **DITA2Go** produces links whenever they make sense: that is, when both source and target are part of the navigation hierarchy for the current project. This vastly simplifies re-use of reltables.

**DITA2Go** silently omits any reltable links where the link target is not included as content in the root map or in any maps referenced from the root map.

## 13.2 Generating and including related links

Which related links **DITA2Go** generates or includes by default depends on the type of link and the source of the link. [Table 13-1](#) shows how each type of link is treated by default. See also §13.3 [Appending links to topics](#) on page 190.

**Table 13-1 Default treatment of related links by link source and type**

Source of related links	Default	Link types
Inferred relationships (not in reltables) in DITA maps	Ignore	previous/next parent/child sibling
Reltables and DITA topic <code>related-links</code> elements	Include	previous/next parent/child sibling
DITA topic <code>related-links</code> elements	Exclude	ancestor/descendant cousin friend

To change the default action for any type of related link:

- [Generate inferred previous/next, parent/child, or sibling links from maps](#)
- [Exclude previous/next, parent/child, or sibling links in topics or reltables](#)
- [Include ancestor/descendant, cousin, or friend links in topics](#)

*Generate inferred  
previous/next,  
parent/child, or  
sibling links from  
maps*

By default, **DITA2Go** ignores previous/next links, parent/child links, and sibling links that exist in DITA maps outside of reltables. To specify which of these links to generate from inferred relationships in maps:

```
[RelatedLinks]
; GeneratePrevNext = No (default) or Yes (generate Previous Topic
; and Next Topic links based on map position)
GeneratePrevNext = Yes
; GenerateParentChild = No (default) or Yes (generate Parent Topic
; and Child Topics links based on map position)
GenerateParentChild = Yes
; GenerateSiblings = No (default) or Yes (generate Related Topic
; links for siblings in the map)
GenerateSiblings = Yes
```

These links are attached to topics based on map position. See also §6.12.4 [Including children of topic headings](#) on page 107.

*Exclude  
previous/next,  
parent/child, or  
sibling links in  
topics or reltables*

By default, **DITA2Go** includes previous/next links, parent/child links, and sibling links that occur in reltables and in DITA topic related-links elements. To exclude any of these links:

```
[RelatedLinks]
; UsePrevNext = Yes (default, use links with role previous or next)
; or No (ignore such links in related-links)
UsePrevNext = No
; UseParent = Yes (default, use links with role parent) or No
UseParent = No
; UseChildren = Yes (default, use links with role child) or No
UseChildren = No
; UseSiblings = Yes (default, use links with role sibling) or No
UseSiblings = No
```

See also §6.12.4 [Including children of topic headings](#) on page 107.

*Include  
ancestor/descendant,  
cousin, or  
friend links in  
topics*

To include ancestor/descendant, cousin, and friend links that occur in DITA topic related-links elements:

```
[RelatedLinks]
; UseAncestors = No (default, ignore links with role ancestor)
; or Yes (use such links)
UseAncestors = Yes
; UseDescendants = No (default, ignore links with role descendant)
; or Yes (use such links)
UseDescendants = Yes
; UseCousins = No (default, ignore links with role cousin) or Yes
UseCousins = Yes
; UseFriends = No (default, ignore links with role friend) or Yes
UseFriends = Yes
```

## 13.3 Appending links to topics

On output you can have **DITA2Go** append to topics either related links or fixed links, or both.

*In this section:*

§13.3.1 [Appending related links to topics](#) on page 191

§13.3.2 [Appending fixed links to topics](#) on page 191

*See also:*

§13.6.3 [Specifying output formats for related and fixed links](#) on page 194

### 13.3.1 Appending related links to topics

By default, on output **DITA2Go** appends to the end of each topic the content of any included `related-links` and `reliable` elements, and any related links generated from maps. To omit these items:

```
[RelatedLinks]
; AppendLinksToTopics = Yes (default, add related-links and reliable
; links at the end of the source topics), or No (omit related topics)
AppendLinksToTopics = No
```

By default, **DITA2Go** divides lists of related topics by topic type, producing separate lists for Concepts, Tasks, and References, plus a Topics list for all others. To prevent list separation:

```
[RelatedLinks]
; UseTopicTypes = Yes (default, separate related topics lists by
; type: Concepts, Tasks, References, or Topics for others), or No
UseTopicTypes = No
```

See also §13.2 [Generating and including related links](#) on page 189.

### 13.3.2 Appending fixed links to topics

After (or instead of) related-link items, you can have **DITA2Go** append fixed links to generated lists; by default **DITA2Go** omits these links:

```
[RelatedLinks]
; Fixed links to include at the bottom after all related links items:
; AddContentsLink = No (default) or Yes
AddContentsLink = Yes
; AddLOFLink = No (default) or Yes
AddLOFLink = Yes
; AddLOTLINK = No (default) or Yes
AddLOTLINK = Yes
; AddGlossaryLink = No (default) or Yes
AddGlossaryLink = Yes
; AddIndexLink = No (default) or Yes
AddIndexLink = Yes
```

The content of each fixed link is the text setting for the heading in question, found in the language configuration file; see §8.9 [Localizing output headings, labels, and names](#) on page 157. For example, in `d2g_lang_en.ini`:

```
[ContentsText]
TOCTitle = Contents

[IndexText]
IDXTITLE = Index
```

See also:

§14.3 [Generating a table of contents](#) on page 198

§14.8 [Producing an index](#) on page 206

## 13.4 Including descriptions with related links

By default, **DITA2Go** does not include descriptions with links. To include text from map `desc`, `shortdesc`, or `linktext` elements:

```
[RelatedLinks]
; UseRelDescription = No (default, do not include any descriptions)
; or Yes (include description from map shortdesc, linktext, or desc)
UseRelDescription = Yes
```

To include `shortdesc` content in the `title` attribute of the heading it applies to, instead of as a paragraph following the heading:

```
[RelatedLinks]
; UseRelDescAsTitle = No (default, treat descriptions as text if used)
; or Yes (use descriptions without formatting as title attributes)
UseRelDescAsTitle = Yes
```

When `UseRelDescAsTitle=Yes`, the `shortdesc` text appears with a mouseover in HTML output. This setting is effective only if you also set `UseRelDescription=Yes`.

**Note:** For the topic itself, the `shortdesc` always appears after the title, unless you specify otherwise; see §6.12.2 [Including shortdesc content in the title attribute](#) on page 106.

*See also:*

§6.13 [Deciding where to display title and shortdesc](#) on page 107

## 13.5 Generating associative links for Help output

To generate associative links (see §16.6.2 [Understanding how ALinks work](#) on page 259) for topics associated by related-links or by reltables:

```
[RelatedLinks]
; GenerateALinks = No (default) or Yes (generate ALinks for topics
; associated by related-links items or by reltables)
GenerateALinks = Yes
```

In a reltable, each row is associated with an ALink name that is applied to all local topics within the row. And each related-links section is also associated with an ALink name.

In either case, you specify the name as the `id` attribute of the `relrow` or the related-links element. Such IDs need to be unique only within their topic or map file, not in the whole project. So you can use the same term for rows in different maps, and in different topics, to identify the ALink groups. **DITA2Go** automatically eliminates any duplicates.

If a `relrow`, or a related-links element, has no `id` attribute, **DITA2Go** skips that row or element. The `id` attribute can have only one value, not a list of values, so you have to live with that limitation when you direct **DITA2Go** to generate ALinks from related-links or reltables.

For Help output, there are other ways to specify ALinks, by inserting **ALink** and **ALinkJump** PI markers in your DITA XML files; see §16.6 [Providing related-topic links for Help systems](#) on page 258. You should choose between allowing **DITA2Go** to generate ALinks and inserting them yourself with PI markers. Do not try to mix the two methods.

## 13.6 Formatting links in output

*In this section:*

§13.6.1 [Understanding how DITA2Go presents related links](#) on page 193

§13.6.2 [Labeling related links](#) on page 193

§13.6.3 [Specifying output formats for related and fixed links](#) on page 194

§13.6.4 [Inserting dividers between topics and lists of links](#) on page 195

### 13.6.1 Understanding how DITA2Go presents related links

By default, **DITA2Go** presents links to related topics three ways, depending on the type of relationship:

[Parent/Child](#)

[Next/Previous](#)

[Sibling](#)

*Parent/Child* For parent/child links, based on the relationship in the map and regardless of reltable or related-links settings, the parent is listed flush left like this:

**Parent topic:** *Link to topic*

The child links are indented, with no heading:

*Link to first child*

*Shortdesc of first child*

*Link to second child*

*Shortdesc of second child*

*Next/Previous* For next/previous links, the treatment is similar to that for parent/child links, with different headings:

**Next topic:** *Link to topic*

**Previous topic:** *Link to topic*

*Sibling* For sibling links, including those in the same row of the reltable, **DITA2Go** presents lists with a bold heading flush left, and the links under it, also flush left:

**Related concepts:**

*Link to topic*

**Related tasks:**

*Link to topic*

**Related Reference:**

*Link to topic*

**Related topics:**

*Link to topic*

You can improve on this presentation with CSS. For example:

```
<div>
  <div class="relinfo"><strong>Related concepts</strong><br/>
  <div><a href="DITATestConcept.html"
    title="Overview of Test Suite design.">Test Suite Design</a></div>
</div>
<div class="relinfo"><strong>Related reference</strong><br/>
<div><a href="DITATestReference.html"
  title="DITA features.">Test Suite Coverage</a></div>
</div>
</div>
```

The CSS is just:

```
.relinfo { margin-top: 1em; margin-bottom: 1em }
```

### 13.6.2 Labeling related links

You can include labels with related links. The **Previous**, **Next**, and **Parent** labels are on the same line as the single topic title; the other labels are actually headings on the line above their topic list, which is indented below them, one topic per line.

Text values for labels and headings are provided in each of the language configuration files; see §8.9 [Localizing output headings, labels, and names](#) on page 157. The values shown here are the default values specified in `d2g_lang_en.ini`.

The following settings indicate which types of relationship to show as a labeled group; if a setting is unspecified (= *blank*), that label is omitted, but the topics under it are listed if their “Use” setting (see §13.2 [Generating and including related links](#) on page 189) is Yes.

To specify labels based on type of link, in a language configuration file:

```
[RelatedLinksText]
; PrevHead = label to use for previous topic in sequence
; (on same line)
PrevHead=Previous Topic:
; NextHead = label to use for next topic in sequence (on same line)
NextHead=Next Topic:
; ParentHead = label to use for parent topic (on same line)
ParentHead=Parent Topic:
; ChildHead = label to use for child topics list (on line above)
ChildHead=Child Topics:
; SiblingHead = label to use for siblings
SiblingHead=Related Topics:
; AncestorHead = label to use for role ancestor (on line above)
AncestorHead=Ancestor Topics:
; DescendantHead = label to use for role descendant (on line above)
DescendantHead=Descendant Topics:
; CousinHead = label to use for role cousin (on line above)
CousinHead=Cousin Topics:
; FriendHead = label to use for role friend (on line above)
FriendHead=Friend Topics:
```

To omit a label, set the label value to blank; for example:

```
[RelatedLinksText]
FriendHead =
```

To specify labels based on topic type (the type attribute of the link or topicref):

```
[RelatedLinksText]
; ConceptsHead = label for concept topics
ConceptsHead=Related Concepts:
; TasksHead = label for task topics
TasksHead=Related Tasks:
; ReferencesHead = label for reference topics
ReferencesHead=Related References:
; RelatedHead = label to use for other related topics (on line above);
RelatedHead=Related Topics:
```

### 13.6.3 Specifying output formats for related and fixed links

To specify formats for various types of related links:

```
[RelatedLinks]
; HeadInlineFormat = format for inline heads (Prev, Next, Parent)
HeadInlineFormat = Strong
; HeadBlockFormat = format for list heads (Child, Sib, Related)
HeadBlockFormat = RelatedHead
; TopicTitleFormat = block format for topic title lines (including
; those with inline heads)
TopicTitleFormat = RelatedTopic
; ShortdescFormat = block format to use for link description
ShortdescFormat = RelatedDesc
```

To specify a format for fixed links to generated lists:



```
[RelatedLinks]
; AddedLinksFormat = format to use, default RelatedTopic
AddedLinksFormat = RelatedHead
```

Configure link formats in the appropriate format configuration file; see §7.2.1 [Understanding where to define output formats](#) on page 110.

To configure components of fixed-link entries:

```
[RelatedLinks]
; AddedLinksSpacer = text to put between added links, can include
; <U+nnnn/> for symbols, and <br/> for a line break
AddedLinksSpacer = <spc/><U+2022/><spc/>
; AddedLinksStart = text to put before first link, default none
AddedLinksStart =
; AddedLinksEnd = text to put after last link, default none
AddedLinksEnd =
```

These settings use format-component processing, so can include any format-component properties except `<name/>`, which is meaningless in this context. See §8 [Configuring format components](#) on page 141.

With the default values, if you turn on all fixed-link options, you get a row of links below the related links that looks something like this, where each term is a link to the corresponding generated list:

**[Contents](#) \* [Figures](#) \* [Tables](#) \* [Glossary](#) \* [Index](#)**

See §13.3 [Appending links to topics](#) on page 190.

### 13.6.4 Inserting dividers between topics and lists of links

By default, **DITA2Go** inserts a space as a divider between a topic and its related links.

To omit any divider between topic and links:

```
[RelatedLinks]
; UseRelatedDivider = Yes (default, put divider para before any
; related links) or No
UseRelatedDivider = No
; RelatedDividerFormat = block format for divider, if used
RelatedDividerFormat = RelatedDivider
```

To specify a divider between related links (or the topic, if you are not including related links) and any following fixed links:

```
[RelatedLinks]
; UseAddedDivider = No (default) or Yes (put divider para after any
; related links and before added links)
UseAddedDivider = Yes
; AddedDividerFormat = block format for divider, if used
AddedDividerFormat = RelatedDivider
```

The default value for `AddedDividerFormat` is the same format specified for `RelatedDivider`. Configure divider formats in the appropriate format configuration file; see §7.2.1 [Understanding where to define output formats](#) on page 110.

To specify content for the divider(s), in a language configuration file:

```
[RelatedLinksText]
; RelatedDividerText = content for divider para (default is a space)
RelatedDividerText = ~::~::~
; AddedDividerText = content for divider para (default is a space)
AddedDividerText =
```

See §8.9 [Localizing output headings, labels, and names](#) on page 157.

## 13.7 Changing link path for peer related links

To change the reference for a related link to the actual location when `scope="peer"` (for example):

```
[PeerLinks]
; href of link = path to use in target links
../ditatopic.dita=finaltopic.htm
```

# 14 Generating lists and indexes

---

If the output type for your **DITA2Go** project requires generated lists such as a table of contents, a list of figures, a list of tables, or an index, or multiple versions of these lists, **DITA2Go** can produce them, replete with links. Topics include:

- §14.1 [Understanding how DITA2Go produces lists](#) on page 197
- §14.2 [Naming generated HTML list and index files](#) on page 198
- §14.3 [Generating a table of contents](#) on page 198
- §14.4 [Generating a list of figures](#) on page 201
- §14.5 [Generating a list of tables](#) on page 202
- §14.6 [Treating figure titles as table titles](#) on page 203
- §14.7 [Producing a glossary](#) on page 203
- §14.8 [Producing an index](#) on page 206
- §14.9 [Configuring variant booklist components](#) on page 213

## 14.1 Understanding how DITA2Go produces lists

**DITA2Go** uses bookmap elements that are specified as components of frontmatter and backmatter booklists to determine which lists to produce and what goes into each list. The DITA rule is simple: if the booklist element is present, and has no `@href`, the processor (in this case **DITA2Go**) *must* generate the specified list. If you do not want a particular list, remove the booklist element from the bookmap, or condition it out.

A DITA bookmap can contain several elements that call for generated lists: `toc`, `indexlist`, `figurelist`, `tablelist`, `glossarylist`, `bibliolist`, `abbrevlist`, `trademarklist`, `amendments`, and `booklist`; the last is a general case for specialization. All are derived from `topicref`. By default, **DITA2Go** assigns element type properties `List`, `Ref`, `Topic`, and `Map` to each of these bookmap elements; see §11.2 [Specifying properties of element types](#) on page 179. To produce multiple variations of any of these booklist types, see §14.9 [Configuring variant booklist components](#) on page 213.

Some bookmap list elements may be authored directly, such as `bibliolist`, `abbrevlist`, `trademarklist`, and `amendments`. **DITA2Go** handles `toc` via maps and `indexlist` via `indexterms`. A `glossarylist` can be produced by a map of `glossentry` topics. However, `figurelist` and `tablelist` (and maybe `abbrevlist` and `trademarklist`) must be generated from topics. To produce lists of tables and figures, **DITA2Go** collects copies of table and figure titles, applies a user-specifiable format for the list, and adds top and bottom information.

If a DITA map element has a `navtitle`, **DITA2Go** uses that title in place of any user-specified title for a given list. If the `navtitle` includes an `@href`, **DITA2Go** assumes the link points to a valid topic or map file, and therefore does not generate a list, but just processes the referenced file.

Elements with content to be included as entries in lists or indexes must have `id` attributes. You can omit an element from a list by eliminating its `id` attribute.

*See also:*

- §27.2.1 [Choosing between splitting and chunking](#) on page 523

## 14.2 Naming generated HTML list and index files

For HTML output, you can specify base names and identifying suffixes for generated list and index files. However, unless you need to match an external convention, it is best to use the default values.

By default, all generated list and index files for a project get the same base name, followed by a three-letter suffix that identifies the file type, followed by a file extension that is the value of `[Setup]FileSuffix` specified for your project (see §4.1.6 [Checking output type and file extension](#) on page 70). [Table 14-1](#) lists the generated file types and their default suffixes, and shows the keywords available for specifying alternate values for base name or suffix.

**Table 14-1: Default file name suffixes for generated files**

Generated file	Default suffix	Configuration section	Base name keyword	Suffix keyword	Ref.
Table of contents	TOC	[Contents]	TOCFile	TOCSuffix	<a href="#">14.3.1</a>
List of figures	LOF	[ListOfFigures]	LOFFile	LOFSuffix	<a href="#">14.4</a>
List of tables	LOT	[ListOfTables]	LOTFile	LOTSuffix	<a href="#">14.5</a>
Index	IDX	[Index]	IDXFile	IDXSuffix	<a href="#">14.8.6.2</a>
Glossary	GLS	[Glossary]	GLSFile	GLSSuffix	<a href="#">14.7</a>

If you specify a value for one of the base-name keywords listed in [Table 14-1](#), **DITA2Go** ignores the value of the corresponding suffix keyword, and just uses the base name as specified, followed by the extension. If you specify a base name, no suffix is added.

The default base name for generated files depends on whether you are running a conversion from the **DITA2Go** Project Manager, or from a command prompt, using the command-line version of **DITA2Go**:

[Via DITA2Go Project Manager](#)

[Via DITA2Go DCL.](#)

*Via **DITA2Go**  
Project Manager*

If you are using the **DITA2Go** Project Manager, the default base name for generated files is the base name of the top-level map file in your project. For example, if you set neither `TOCFile` nor `TOCSuffix`, the default file name for the table of contents generated from `mydoc.bookmap` would be `mydocTOC.htm`.

*Via **DITA2Go**  
DCL*

If you are running **DITA2Go** DCL from a command line, the default base name is the base name of the value specified for the `-o` argument; see §2.7.2 [Understanding how to run DITA2Go DCL](#) on page 47. For example, for the following command:

```
dcl -f html -o somename.htm mydoc.bookmap
```

the default name of the table of contents file would be `somenameTOC.htm`.

If you omit the `-o` argument, the default base name is the name of the map file. For example, for the following command:

```
dcl -f html mydoc.bookmap
```

the default name of the TOC file would be `mydocTOC.htm`, just as it would be if you were using the **DITA2Go** Project Manager.

## 14.3 Generating a table of contents

**DITA2Go** produces a table of contents from a DITA bookmap or a ditamap file, following DITA rules. However, if you are creating a Help system such as HTML Help or Oracle

Help for Java, a table of contents is produced automatically by the Help system tool; so for Help systems you do not really need another TOC, unless you intend to use it as the default start page of your output.

*In this section:*

- §14.3.1 [Specifying a file name and title for the TOC](#) on page 199
- §14.3.2 [Deciding what to include in the TOC](#) on page 199
- §14.3.3 [Specifying formats for TOC title, entries, and references](#) on page 200
- §14.3.4 [Including navigation titles from maps in the TOC](#) on page 200

*See also:*

- §14.9 [Configuring variant booklist components](#) on page 213
- §16.3 [Producing contents and index for Help systems](#) on page 248

### 14.3.1 Specifying a file name and title for the TOC

If your project requires producing more than one TOC, you can override the settings described in this section with equivalent settings specific to each variant TOC; see §14.9 [Configuring variant booklist components](#) on page 213.

For HTML output, **DITA2Go** creates a separate file for the TOC; you can specify a file name, or you can simply allow **DITA2Go** to name the file after the map from which the TOC is generated, with suffix TOC. For RTF output, no separate file is produced.

*TOC file name* To specify a file name for the table of contents (HTML output only):

```
[Contents]
; TOCFile = base name to use for output TOC file, with suffix TOC
TOCFile = mapnameTOC
; TOCSuffix = suffix for the base file name, default TOC
TOCSuffix = TOC
```

If you specify a value for TOCFile, **DITA2Go** ignores TOCSuffix. See §14.2 [Naming generated HTML list and index files](#) on page 198. To keep the TOC with any material that precedes it in output, see §27.2.3 [Providing a page break between title and TOC](#) on page 525.

*TOC title* To specify the text of a title for the table of contents, in the language configuration file referenced by your project:

```
[ContentsText]
; TOCTitle = title text for TOC
TOCTitle = Table of Contents
```

Each **DITA2Go**-supplied language configuration file includes a setting for TOCTitle; see §8.9 [Localizing output headings, labels, and names](#) on page 157. The value shown here is the default value specified in `d2g_lang_en.ini`. The `<toc> @navtitle` overrides the value of TOCTitle.

### 14.3.2 Deciding what to include in the TOC

By default, **DITA2Go** includes all `topicref` titles in the TOC, except the `topicref` for the TOC itself.

To also include TOC entries for nested topics:

```
[Contents]
; UseNestedTopicsInTOC = No (default) or Yes (add them
; as though they were specified in map topicrefs)
UseNestedTopicsInTOC = Yes
```

To also include shortdesc content in TOCs generated from maps:

```
[Contents]
; UseTOCDescriptions = No (default, omit shortdesc from TOC) or Yes.
UseTOCDescriptions = Yes
```

When UseTOCDescriptions=Yes, **DITA2Go** includes in the generated TOC all text material from the map. For HTML output, the shortdesc content appears in a mouseover. For RTF output, it seriously clutters the TOC, especially for large documents.

The values of UseNestedTopicsInTOC and UseTOCDescriptions apply to all TOCs produced in your current project.

*See also:*

§6.12.1 [Providing default output formats for map content](#) on page 105

§6.13 [Deciding where to display title and shortdesc](#) on page 107

### 14.3.3 Specifying formats for TOC title, entries, and references

If your project produces more than one TOC, you can override the settings in this section with equivalent settings specific to each TOC; see §14.9 [Configuring variant booklist components](#) on page 213.

To specify formats for TOC title and cross references:

```
[Contents]
; TOCFormat = format to use for TOC entries
TOCFormat = TOC1
; TOCTitleFormat = format for TOC title
TOCTitleFormat = ContentsTitle
; TOCXrefFormat = xref subformat for TOC xrefs to topics.
TOCXrefFormat = TOCXref
```

The format for topicrefs, which turn into TOC entries, is the base map format name followed by a sequence number; see §6.12.1 [Providing default output formats for map content](#) on page 105. These formats are already mapped to a fairly deep level in configuration file d2g\_config.ini, located in %OMSYSHOME%\d2g\configs. If you change any format names, you must either alias your names to the default names, or provide your own definitions in a format configuration file; see §7 [Configuring output formats](#) on page 109.

### 14.3.4 Including navigation titles from maps in the TOC

The <navtitle> element, as opposed to the @navtitle attribute, is supported in maps beginning with DITA version 1.2. To direct **DITA2Go** to look for <navtitle> elements instead of navtitle attributes in <topichead>s, to use for TOC entries:

```
[TopicHeads]
; TopicheadsHaveNavtitles = No (default, treat topicheads as
; topics only if they have a navtitle attribute, not element)
; or Yes (assume all topicheads have navtitles)
TopicheadsHaveNavtitles = Yes
```

To direct **DITA2Go** to treat the value of locktitle as “yes” when this attribute is missing:

```
[MapOptions]
; LockAllNavtitles = No (default, require @locktitle="yes" to
; use navtitles) or Yes (make missing @locktitle default to yes)
LockAllNavtitles = Yes
```

To include all topics in the TOC by default:

```
[MapOptions]
; UseAllInTOC = No (default, require @toc="yes" to put topic
; in TOC), or Yes (put all topics in TOC unless @toc="no")
UseAllInTOC = Yes
```

If you are producing a Help output type, also see §16.4.2 [Including contents entries in HTML-based Help](#) on page 250.

## 14.4 Generating a list of figures

**DITA2Go** produces a list of figures when your DITA source includes a bookmap figurelist element. **DITA2Go** includes in the list of figures the title element in each fig element or element derived from fig (unless you assign to it element types that do not include fig), provided the fig element has an id attribute. Requiring the id attribute gives you a way to exclude items from the list: **DITA2Go** omits the titles of fig elements that have no @id.

**DITA2Go** supports title elements in imagemap elements, despite the DITA prohibition against this. If you put titles in image maps, those titles can be included in the list of figures. The DITA topic file containing the image map will no longer be valid DITA, but it will be well formed DITA.

If your project includes more than one <figurelist>, you can override the settings in this section with equivalent settings specific to each list of figures; see §14.9 [Configuring variant booklist components](#) on page 213.

The properties you can specify for a list of figures include:

[File name and suffix](#)

[Text of the list title](#)

[Output formats.](#)

*File name and  
suffix*

For HTML output, **DITA2Go** creates a separate file for a list of figures; you can specify a file name, or you can simply allow **DITA2Go** to name the file after the map from which the list of figures is generated, with suffix LOF. For RTF output, no separate file is produced.

To specify a file name for the list of figures (HTML output only):

```
[ListOfFigures]
; LOFFile = base name to use for output file, plus suffix LOF
LOFFile = mapnameLOF
; LOFSuffix = suffix for the base file name, default LOF
LOFSuffix = LOF
```

If you specify a value for LOFFile, LOFSuffix is ignored. See §14.2 [Naming generated HTML list and index files](#) on page 198.

*Text of the list title*

To specify text for the title of the list of figures, in the language configuration file referenced by your project:

```
[ListOfFiguresText]
; LOFTitle = title text for list of figures
LOFTitle = List of Figures
```

Each **DITA2Go**-supplied language configuration file includes a setting for LOFTitle; see §8.9 [Localizing output headings, labels, and names](#) on page 157. The value shown here is the default value specified in d2g\_lang\_en.ini.

*Output formats*

To specify formats for the list of figures:

```
[ListOfFigures]
; LOFTitleFormat = format for LOF title
```



```

LOFTitleFormat = FigureListTitle
; LOFFormat = default format to use for LOF entries
LOFFormat = FigureListItem
; LOFTOCFormat = format for TOC entry for list of figures title
LOFTOCFormat = LOF
; LOFXrefFormat = format for cross references to LOF
LOFXrefFormat = FigureTitleXref

```

All values here are the default format names. If you change any format names, you must either alias your names to the default names, or provide your own definitions in a format configuration file; see §7 [Configuring output formats](#) on page 109.

## 14.5 Generating a list of tables

**DITA2Go** produces a list of tables when your DITA source includes a `bookmap` `tablelist` element. **DITA2Go** includes in the list of tables the `title` element under each `table` element or element derived from `table` (unless you assign to it element types that do not include `table`), provided the `table` element has an `id` attribute. Requiring the `id` attribute gives you a way to exclude items from the list: **DITA2Go** omits the titles of `table` elements that have no `@id`.

If your project includes more than one `<tablelist>`, you can override the settings in this section with equivalent settings specific to each list of tables; see §14.9 [Configuring variant booklist components](#) on page 213.

The properties you can specify for a list of tables include:

[File name and suffix](#)

[Text of the list title](#)

[Output formats.](#)

*File name and  
suffix*

For HTML output, **DITA2Go** creates a separate file for a list of tables; you can specify a file name, or you can simply allow **DITA2Go** to name the file after the map from which the list of tables is generated, with suffix `LOT`. For RTF output, no separate file is produced.

To specify a file name for the list of tables (HTML output only):

```

[ListOfTables]
; LOTFile = base name to use for output file, plus suffix LOT
LOTFile = mapnameLOT
; LOTSuffix = suffix for the base file name, default LOT
LOTSuffix = LOT

```

If you specify a value for `LOTFile`, `LOTSuffix` is ignored. See §14.2 [Naming generated HTML list and index files](#) on page 198.

*Text of the list title*

To specify a title for the list of tables, in the language configuration file referenced by your project:

```

[ListOfTablesText]
; LOTTitle = title text for list of tables
LOTTitle = List of Tables

```

Each **DITA2Go**-supplied language configuration file includes a default setting for `LOTTitle`; see §8.9 [Localizing output headings, labels, and names](#) on page 157. The value shown here is the default value specified in `d2g_lang_en.ini`.

*Output formats*

To specify formats for the list of tables:

```

[ListOfTables]
; LOTTitleFormat = format for LOT title
LOTTitleFormat = TableListTitle

```



```

; LOTFormat = default format to use for LOT entries
LOTFormat = TableListItem
; LOTTOCFormat = format for TOC entry for list of tables title
LOTTOCFormat = LOT
; LOTXrefFormat = format for cross references to LOT
LOTXrefFormat = TableTitleXref

```

All values here are the default format names, specified in configuration template `d2g_config.ini`, located in directory `%OMSYSHOME%\d2g\config`. If you change any format names, you must either alias your names to the default names, or provide your own definitions in a format configuration file; see §7 [Configuring output formats](#) on page 109.

## 14.6 Treating figure titles as table titles

If you use a `fig` element to hold a `<simpletable>` in order to give the `<simpletable>` a title, you might want the `fig` title included in the list of tables instead of in the list of figures. To make this happen, give either the `fig` element or its title an `@outputclass` that contains the string “table” or “Table”. By default, **DITA2Go** includes such titles in the list of tables instead of the list of figures; and applies the table-title output format instead of the figure-title output format.

To instead keep titles of `fig` elements that include an `@outputclass` with the string “table” or “Table” in the list of figures:

```

[FigureOptions]
; TreatTableFigAsTable = Yes (default), or No
TreatTableFigAsTable = No

```

When `TreatTableFigAsTable=Yes`, if either a `fig` element or its title element has an `@outputclass` that contains the string “table” or “Table”, the figure title gets the table title format and goes into the list of tables instead of the list of figures.

For example:

```

<fig id="figspec">
<title outputclass="TableTitle">Figure Containing a Table</title>
<simpletable>
...
</simpletable>
</fig>

```

The title *Figure Containing a Table* gets the table title format, because of the `@outputclass`, and also goes into the list of tables instead of into the list of figures.

## 14.7 Producing a glossary

**DITA2Go** provides special processing for cross references to glossary items; terms and keywords that use keyrefs to get to a glossary item; and support for `<abbreviated-form>`. You can specify a title for the glossary, and configure output formats and use of abbreviations. For HTML output you can specify a file name.

*In this section:*

- §14.7.1 [Specifying file name and title for the glossary](#) on page 204
- §14.7.2 [Specifying output formats for the glossary](#) on page 204
- §14.7.3 [Configuring use of abbreviations for glossary terms](#) on page 205

*See also:*

- §22.9 [Providing hover text for links in HTML](#) on page 441

## 14.7.1 Specifying file name and title for the glossary

To specify a file name for the glossary (HTML output only):

```
[Glossary]
; GLSFile = base name to use for output file, plus suffix GLS
GLSFile = mapnameGLS
; GLSSuffix = suffix for the base file name, default GLS
GLSSuffix = GLS
```

If you specify a value for GLSFile, GLSSuffix is ignored. See §14.2 [Naming generated HTML list and index files](#) on page 198.

To specify text for the glossary title, in the language configuration file referenced by your project:

```
[GlossaryText]
; GLSTitle = title text for glossary
GLSTitle = Glossary
```

Each **DITA2Go**-supplied language configuration file includes a setting for GLSTitle; see §8.9 [Localizing output headings, labels, and names](#) on page 157. The value shown here is the default value specified in `d2g_lang_en.ini`.

## 14.7.2 Specifying output formats for the glossary

To specify output formats for the glossary:

```
[Glossary]
; GLSTitleFormat = format for the glossary title
GLSTitleFormat = GlossaryTitle
; GLSTOCFormat = format for TOC entry for the glossary title
GLSTOCFormat = GLS
```

Default formats for glossary components and references are specified in

`%OMSYSHOME%\d2g\system\config\d2g_config.ini`:

```
[BlockFormatMaps]
glossarylist/* = GLS
glossarylist/booklists/backmatter/bookmap/1 = TOC2P
title/$topic/glossarylist/booklists/backmatter/$map/1 = GlossaryTitle
glossterm/* = Heading1
glossdef/* = Abstract
glossAbbreviation/* = Heading2S
glossAcronym/* = Heading2S
glossSynonym/* = Heading2S
gloss* = Body

[InlineFormatMaps]
glossPartOfSpeech/* = Bold
gloss* = No
```

See §6.4.2 [Mapping block and inline element paths to formats](#) on page 93. The formats are defined in `%OMSYSHOME%\d2g\system\formats\d2htm_formats.ini` (for HTML output) and `d2rtf_formats.ini` (for RTF output).

If you change any format names, you must either alias your names to the default names, or provide your own definitions in a format configuration file; see §7 [Configuring output formats](#) on page 109.

See also §14.7.3.4 [Formatting output for the abbreviated-form element](#) on page 206.

### 14.7.3 Configuring use of abbreviations for glossary terms

Your company style guide might require certain terms to be spelled out in full the first time they appear in your document, after which you can use an abbreviation or acronym. This can present some challenges:

- If such a term is included in a glossary, you must be able to represent the relationship between either form of the term and its definition.
- In a reusable-topic-based writing environment there is no way to know which will be the first use of a term.
- The first-use rule can vary depending on the context, on the type of output, and other possible factors.

In DITA 1.2 and later versions, you can use the `<abbreviated-form>` element to represent every reference to a glossary term.

*In this section:*

§14.7.3.1 [Specifying first-use rules for glossary-term abbreviations](#) on page 205

§14.7.3.2 [Overriding first-use rules for abbreviations](#) on page 206

§14.7.3.3 [Specifying @class for the abbreviated-form element](#) on page 206

§14.7.3.4 [Formatting output for the abbreviated-form element](#) on page 206

#### 14.7.3.1 Specifying first-use rules for glossary-term abbreviations

**DITA2Go** determines whether to output the full form or the abbreviated form, based in part on where the reference occurs:

[DITAmaps \(including bookmaps\) and their submaps](#)

[Topic title elements.](#)

*DITAmaps  
(including  
bookmaps) and  
their submaps*

In maps and submaps, at the first occurrence of:

```
<abbreviated-form keyref="key" />
```

where *key* is the value defined on the `keys` attribute of the `<glossref>` element that references the `<glossentry>` topic for the term, **DITA2Go** includes the content of the `<glossSurfaceForm>` element for the term. For all subsequent occurrences in the same topic, **DITA2Go** includes the abbreviated form instead: that is, the content of the `<glossAcronym>` element for the term.

**DITA2Go** provides a way for you to specify the map level where the first-use rule should apply. For example, to show the full term the first time it is used in a `<chapter>`:

```
[Glossary]
; ResetAbbrevAt = Level in bookmap to restart (put out
; glossSurfaceForm again). One of: Document, Part, Chapter,
; Head1, ..., Head6, or Topic; default is Topic.
ResetAbbrevAt = Chapter
```

The default value of `ResetAbbrevAt` is `Topic`.

**DITA2Go** treats appendix as equivalent to Chapter. Part and Chapter apply only to bookmaps. In a bookmap, the `Head1` through `Head6` levels start under Chapter.

`Head1` through `Head6` apply to the titles of topics, not to section titles within topics.

The first occurrence of `<abbreviated-form>` at the level specified by `ResetAbbrevAt` gets the full term. All subsequent occurrences at the same level, and at lower levels in the same map and its submaps, get the abbreviated form.

*Topic title  
elements*

To specify whether terms in titles should be treated differently from terms in text with respect to the first-use rule:

```
[Glossary]
; UseAbbrevInTitles = Yes (default, use glossAcronym,
; not glossSurfaceForm, in title elements) or No
; (treat title instances the same as any others).
UseAbbrevInTitles=Yes
```

When `UseAbbrevInTitles=Yes`, every instance of `<abbreviated-form>` in a topic `<title>` gets the content of `<glossAcronym>`. However, if it is the instance that would be considered first use according to the value of `ResetAbbrevAt`, the next occurrence in that topic of `<abbreviated-form>` with the same key gets the content of `<glossSurfaceForm>`.

When `UseAbbrevInTitles=No`, title elements are treated the same as any other elements with respect to first use of `<abbreviated-form>`.

### 14.7.3.2 Overriding first-use rules for abbreviations

**DITA2Go** determines the identity of a reference in `<abbreviated-form>` based on the `@key`, not on what the `@keydef` resolves to. To force a restart you could put two or more keys in the `@keydef`, and move on to the next in the `@keyref`.

### 14.7.3.3 Specifying @class for the abbreviated-form element

The DITA version 1.2 DTDs seem to have a deficiency that prevents them from adding the `@class` to `<abbreviated-form>`, which makes it not work. To overcome this deficiency:

```
[ElementClasses]
abbreviated-form=+ topic/term abbrev-d/abbreviated-form
```

**Note:** The value has a trailing space.

See §11.2.4 [Adding the full class attribute to an element](#) on page 183. Perhaps this has been fixed in a newer version of the DTDs.

### 14.7.3.4 Formatting output for the abbreviated-form element

When **DITA2Go** encounters `<abbreviated-form>` in your DITA source, in addition to displaying the correct form of the referenced term based on first-use rules and settings, the rendered term becomes an active cross reference to the glossary item, and the `<glossTerm>` content (title) shows on mouseover; see §22.9 [Providing hover text for links in HTML](#) on page 441.

If you specify `@outputclass` on `<abbreviated-form>`, **DITA2Go** treats the value as the name of a cross-reference output format; otherwise, **DITA2Go** uses cross-reference format `TextXref`. See §8.7 [Defining cross-reference output formats](#) on page 155.

## 14.8 Producing an index

**DITA2Go** produces a customizable index for HTML or print RTF output, based on `<index*>` elements in your DITA document. **DITA2Go** generates indexes by default when you convert a bookmap that includes `indexlist` elements.

*In this section:*

- §14.8.1 [Specifying output formats for the index](#) on page 207
- §14.8.2 [Overriding formats for index entries and references](#) on page 207
- §14.8.3 [Configuring see and see-also index entries](#) on page 208
- §14.8.4 [Configuring index references](#) on page 209

§14.8.5 [Including heading letters in the index](#) on page 210

§14.8.6 [Configuring index features for HTML output](#) on page 211

See also:

§14.9 [Configuring variant booklist components](#) on page 213

§16.3 [Producing contents and index for Help systems](#) on page 248

## 14.8.1 Specifying output formats for the index

If your project includes more than one index, you can override the settings described here with equivalent settings specific to each index; see §14.9 [Configuring variant booklist components](#) on page 213. The values shown here are default values for all indexes.

To specify output formats for index title, entries, and TOC entry:

```
[Index]
; IDXFormat = default format for index entries
IDXFormat = Index1
; IDXTitleFormat = format for index title
IDXTitleFormat = IndexTitle
; IDXTOCFormat = format of TOC entry for index title
IDXTOCFormat = IndexTOC
```

**DITA2Go** uses the value of `IDXFormat` *only* if all of the following are true:

- The required item-level format is missing from both:
  - `[IndexEntryFormats]` (see §14.8.2 [Overriding formats for index entries and references](#) on page 207) and
  - `[VariantNameBLForms]` (see §14.9.4 [Defining properties of items in variant booklist components](#) on page 216).
- No value is specified for `[VariantNameBList]ItemFormat` (see §14.9.3 [Specifying properties of variant booklist components](#) on page 215).

By default, **DITA2Go** writes the index title as a heading at the top of the index page. For HTML output only, you can omit this heading:

```
[Index]
; UseIndexHeading = Yes (default, put a title on the index) or No.
UseIndexHeading = No
```

When `UseIndexHeading=Yes`, the title appears at the top of the index page. To provide a different kind of title of your own devising, set `UseIndexHeading=No` and use `[Inserts]` keyword `IndexTop` to specify code or a macro to produce a title; see §14.8.6.4 [Customizing and linking to the index file](#) on page 213.

To specify the text of a title for the index, in the language configuration file referenced by your project:

```
[IndexText]
; IDXTitle = text of index title
IDXTitle = Index
```

**DITA2Go** uses the value of `IDXTitle` as the index title if the `indexlist` element does not contain a `@navtitle`, and also uses this value for references to the index from the TOC and from related-links lists.

## 14.8.2 Overriding formats for index entries and references

You do not need to assign formats to index-entry levels, except to override the default assignments. The default formats assigned to index-entry levels 1 through 9 are defined in system format configuration files.

To assign level-specific formats for compact index entries:

```
[IndexEntryFormats]
; level number = format
1 = Index1
...
9 = Index9
```

To assign level-specific formats for full-title index references (not needed for the compact form):

```
[IndexRefParaFormats]
; level number = format (used only for non-compact indexes)
1 = IndexRef1
...
9 = IndexRef9
```

See §14.8.4 [Configuring index references](#) on page 209.

### 14.8.3 Configuring *see* and *see-also* index entries

To determine how *see* and *see-also* entries are displayed in the index, you can specify:

[Format component names](#)

[Component building blocks](#)

[Text for redirection entries.](#)

*Format  
component  
names*

To specify output formats for index *see* and *see-also* entries:

```
[IndexSeeFormats]
; IndexSeeStart = format to use at start of index-see references
IndexSeeStart = SeeStartIndex
; IndexSeeAlsoStart = format to use at start of index-see-also
IndexSeeAlsoStart = SeeAlsoStartIndex
; IndexSeeEnd = format to use at end of index-see
IndexSeeEnd = SeeEndIndex
; IndexSeeAlsoEnd = format to use at end of index-see-also
IndexSeeAlsoEnd = SeeAlsoEndIndex
```

All values here are the default format names. If you change any format names, you must either alias your names to the default names, or provide your own definitions in a subformat configuration file.

*Component  
building blocks*

You can use the building blocks described in §8.1.5 [Including typographic tags and character formats](#) on page 143 to fine-tune index *see* and *see-also* entries:

```
[SeeStartIndex]
help = Start of "see" index entry
form = <spc/><i><name/></i><spc/>

[SeeAlsoStartIndex]
help = Start of "see also" index entry
form = <i><name/></i><spc/>form = :<i><name/></i><spc/>

[SeeEndIndex]
help = End of "see" index entry
form =

[SeeAlsoEndIndex]
help = End of "see also" index entry
form =
```

These are the default formats, defined in Subformats configuration file `d2htm_subformats.ini`.

*Text for  
redirection entries*

To specify text for index *see* and *see-also* entries, in the language configuration file referenced by your project:

```
[IndexSeeText]
; SeeStartIndex = content to use at start of index-see entries
SeeStartIndex = see
; SeeAlsoStartIndex = content to use at start of index-see-also
SeeAlsoStartIndex = See also
; SeeEndIndex = content to use at end of index-see
SeeEndIndex =
; SeeAlsoEndIndex = content to use at end of index-see-also
SeeAlsoEndIndex =
```

Each value becomes the <name/> content of the corresponding format component. A colon at the end of the value specified for SeeEndIndex or SeeAlsoEndIndex forces the following item to the next index level.

Text for the settings in [IndexSeeText] is specified in language configuration files; see §8.9 [Localizing output headings, labels, and names](#) on page 157. The text shown here for these settings is the default text specified in d2g\_lang\_en.ini, located in directory %OMSYSHOME%\d2g\system\lang.

### 14.8.4 Configuring index references

For cross references from index entries to topics, you can choose between displaying links as stacked topic titles and displaying them as a series of references (usually page numbers) in the same paragraph as the index entry.

If your project includes more than one index, you can override the settings described here with equivalent settings specific to each index; see §14.9 [Configuring variant booklist components](#) on page 213.

To specify how to display index references:

```
[Index]
; UseCompactForm = Yes (default; icons for HTML, page numbers for
; RTF), or No (topic titles, one per line, for either HTML or RTF)
UseCompactForm = Yes
; FullIndexRanges = Yes (default for RTF) or No (default for HTML)
FullIndexRanges = No
```

When UseCompactForm=Yes, all references from an index entry are added to the end of the index-entry paragraph:

```
text of index entry  5, 12, 21-26, 40, ...
```

This is the default presentation for RTF output. For HTML output, the page numbers are replaced by the symbol ✱, and only the start member of a start/end <indexterm> pair is included (unless you set FullIndexRanges=Yes). To use a different symbol, override the Unicode value in cross-reference format [IndexIconXref]; see §8.7 [Defining cross-reference output formats](#) on page 155.

When FullIndexRanges=Yes, both members of a start/end <indexterm> pair are included in the index; this is the default for RTF output when UseCompactForm=Yes.

When FullIndexRanges=No, only the start member is included in the index, this is the default for HTML output when UseCompactForm=Yes.

When UseCompactForm=No, each reference shows the full text of the topic title to which the index entry refers, and multiple references from each index entry are stacked under the entry. Only the start member of any start/end <indexterm> pair is included:

```
text of index entry
  topic1 title
  topic2 title
  ...
```



To specify separators for multiple references, in the language configuration file referenced by your project:

```
[IndexText]
; IndexRefStartSep = separator between entry and first reference
IndexRefStartSep =
; IndexRefSep = separator between multiple references
IndexRefSep = ,
; IndexRangeSep = separator between references in a range
IndexRangeSep = -
```

These are the default values:

- a space between the entry and its first reference
- a comma and a space between successive references
- a dash between start and end references in a range.

For RTF output only, when UseCompactForm=Yes, you can include a leader between each index entry and the first page number for that entry:

```
[Index]
; UseIndexLeader = No (default) or Yes, include a tab with a leader.
UseIndexLeader = Yes
```

The default leader is a row of dots. To use a different leader, you must override the value of property tabs in format [IndexBase]; see §7.4.6 [Modifying DITA2Go default output formats](#) on page 116.

## 14.8.5 Including heading letters in the index

You can include individual letters of the alphabet as headings to introduce each alphabetic section of the index. You can also include letters of the alphabet across the top of the page, each with a link to the corresponding heading letter within the index.

*Heading letters as  
separators*

To include heading letters to introduce each alphabetic section of the index:

```
[Index]
; UseIndexLetters = Yes (default, insert alphabetic) or No
UseIndexLetters = Yes
; IndexLetterSymbol = text to use, or blank to omit
IndexLetterSymbol = Sym
; IndexLetterNumber = text to use, or blank to omit
IndexLetterNumber = Num
; IndexLettersFormat = format to use for index alphabet headings
IndexLettersFormat = IndexLetters
; IndexLetterPrefix = prefix to use for anchors for jumps to
; index heading letters
IndexLetterPrefix = ixlet
```

When UseIndexLetters=Yes, **DITA2Go** inserts, as a heading, a letter of the alphabet at the beginning of each section of the index where entries start with that letter.

By default, heading characters are included for entries that start with a digit or with a non-alphanumeric character: for digits, the default heading is Num; and for non-alphanumeric characters, the default heading is Sym. You can change these headings. If you specify a sort order that ignores digits or symbols, you can set IndexLetterNumber or IndexLetterSymbol to blank. See §16.5.8 [Customizing index sort order](#) on page 256.

*Heading letters at  
top of page*

To include a row of letters across the top of each index page, linking to the heading letters:

```
[Index]
; UseIndexTopLetters = Yes (default for RTF; for HTML, default only if
; IndexNavType=HTML: top-of-page alphabetic that link to heading
; letters) or No
```



```

UseIndexTopLetters = Yes
; IndexTopLettersFormat = format to use for top-of-page alphabet
IndexTopLettersFormat = IndexTopLetters

```

When `UseIndexTopLetters=Yes`, **DITA2Go** writes the alphabet across the top of the index page. Each letter is an active link to the same heading letter in the body of the index, unless there are no entries that begin with that letter.

For HTML, `UseIndexTopLetters` is effective only when `IndexNavType=HTML`; see §14.8.6.1 [Choosing the type of index to generate for HTML](#) on page 211.

## 14.8.6 Configuring index features for HTML output

The settings described here are intended for non-Help HTML output. You can also use the settings described for Help systems; see §16.5 [Configuring index entries for Help systems](#) on page 251.

Values of the settings described here become the default values for all HTML indexes produced by a conversion. If you are creating multiple indexes, see §14.9 [Configuring variant booklist components](#) on page 213 for values you can specify individually for each index.

*In this section:*

§14.8.6.1 [Choosing the type of index to generate for HTML](#) on page 211

§14.8.6.2 [Specifying a file name and suffix for the index](#) on page 212

§14.8.6.3 [Specifying CSS classes for index components](#) on page 212

§14.8.6.4 [Customizing and linking to the index file](#) on page 213

*See also:*

§16.5 [Configuring index entries for Help systems](#) on page 251

### 14.8.6.1 Choosing the type of index to generate for HTML

To specify the type of index to generate for HTML or XHTML output:

```

[Index]
; IndexNavType = HTML (default, plain HTML, uses idxhtm.css),
; CSS (pure CSS, uses idxcss.css), or JavaScript (uses idxjs.*)
IndexNavType = HTML
; WriteIndexCssLink = Yes (default, write link for index CSS) or No
WriteIndexCssLink = Yes

```

**DITA2Go** gathers all the index terms from your document into one of three types of multi-level index, depending on the value of `IndexNavType`:

<u>IndexNavType</u>	<u>CSS file</u>	<u>Default effect</u>
HTML	idxhtm.css	All entries visible initially; optional heading letters across the top of the page, with links
CSS	idxcss.css	Only heading letters visible initially, arrayed vertically on the left; mouseover reveals next level down
JavaScript	idxjs.css	Only index letters visible initially, arrayed vertically on the left; click reveals next level down

**DITA2Go** accesses the required CSS (and JavaScript, if `IndexNavType=JavaScript`) from `%OMSYSHOME%\common\local\htmlidx` or, if the files are not there, from `%OMSYSHOME%\common\system\htmlidx`.

You can copy the CSS and JavaScript files from `\system\htmlidx` to `\local\htmlidx` and edit them there, to customize index appearance for your HTML

output. Do not change the names of these files, and do not edit the files in %OMSYSHOME%\common\system\htmlidx.

When IndexNavType=HTML, you can also include heading letters across the top of the page, with links to their counterparts within the index. See §14.8.5 [Including heading letters in the index](#) on page 210.

When WriteIndexCssLink=Yes, **DITA2Go** writes into the index file <head> the link to the extra CSS file that is called for according to the value of IndexNavType. For example, if IndexNavType=HTML, the link would look like this:

```
<link rel="stylesheet" href="idxhtm.css" type="text/css">
```

WriteIndexCssLink=No is intended to let you turn off the added CSS links for the index, in case you have custom CSS that already includes them, or in case you add your own in-line styles.

### 14.8.6.2 Specifying a file name and suffix for the index

To specify a file name for the index:

```
[Index]
; IDXFile = base name to use for output file, with suffix IDX
IDXFile = mapnameIDX
; IDXSuffix = suffix for the base file name, default IDX
IDXSuffix = IDX
; IndexFileSuffix = suffix to add to map name for index file
; if no IDXFile name (deprecated)
IndexFileSuffix = IDX.htm
```

If you specify a value for IDXFile, IDXSuffix is ignored. See §14.2 [Naming generated HTML list and index files](#) on page 198.

For IndexFileSuffix, you must include the file extension as well as a suffix to add to the base map name. The default suffix is IDX.htm. *Use of IndexFileSuffix is deprecated.* If you provide a value for IDXFile, **DITA2Go** ignores IndexFileSuffix.

### 14.8.6.3 Specifying CSS classes for index components

You can specify CSS class names for index entries, for the links from the index into the HTML content, and (though deprecated) for an entry for the index in the table of contents:

```
[Index]
; IndexLevelClass = class to use for index item <ul>s
IndexLevelClass = IndexEntry
; IndexRefClass = class to use for index reference <ul>s
IndexRefClass = IndexRef
; IndexTOCClass = class to use for index entry in TOC (deprecated)
IndexTOCClass = IndexTOC
```

These class names are also the names of the respective default formats for each of these features; see §14.8.1 [Specifying output formats for the index](#) on page 207. In particular, IndexTOCClass is deprecated in favor of IDXTOCFormat.

**Note:** You can alter the formats by overriding their default definitions; see §7.4.6 [Modifying DITA2Go default output formats](#) on page 116.

If you want a different appearance (other than indentation) for index entries at each level:

```
[Index]
; UseIndexLevelNum = No (default) or Yes (include level number
; in IndexLevelClass and IndexRefClass, requires editing CSS).
UseIndexLevelNum = Yes
```

If you set `UseIndexLevelNum=Yes`, you must also edit the appropriate `idx*.css` file in `\local\htmlidx` to include a definition for each level.

The following settings are deprecated in favor of the corresponding format settings:

```
[Index]
; IndexLetterClass = class for index letters on left (deprecated)
IndexLetterClass = IndexLetters
; IndexTopLettersClass = class for index letters at top (deprecated)
IndexTopLettersClass = IndexTopLetters
```

Use `IndexLettersFormat` and `IndexTopLettersFormat` instead; see §14.8.5 [Including heading letters in the index](#) on page 210.

#### 14.8.6.4 Customizing and linking to the index file

To include a link to the index from the TOC, and to provide code to customize index features:

```
[Inserts]
; IndexHead is within the index-file <head> element, after <title>.
; IndexTOC is inserted at the end of the TOC.
; IndexTop is at the top of the index-file body, after any JS link.
; IndexBottom is at the bottom of the index-file body.
```

For example:

```
[Inserts]
IndexTOC=<$IndexTOCEntry>
IndexTop=<$IndexHeader>
```

If you set `[Inserts]IndexTOC` to your own code or macro, **DITA2Go** uses your code, instead of generating built-in code with the value of `IndexTOCClass` (default `IndexTOC`) as class and `IndexFileTitle` (default `Index`) as content. This means your code must specify class and content of the link from the TOC to the index. You can reference the index file with predefined macro variable `$_indexfilename`:

```
[IndexTOCEntry]
; Link from the TOC to the index:
<H2 align="left" class="PartTOC">
<a href="<$$_indexfilename>">Index</a></H2>

[IndexHeader]
; A roll-your-own title for the index (see d2html_macros.ini):
<H2 align="left" class="IndexHeaderClass">Index</H2>
<br /><br />
```

For other index specifications **DITA2Go** provides `Use*` settings, so if (for example) you want your own heading to replace the one **DITA2Go** generates, you would set `UseIndexHeading=No` and assign code or a macro to `[Inserts]IndexTop`.

See also:

§14.9.5 [Mapping indexterms to variant indexes](#) on page 217

§27.6.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 535

§37.3.4 [Using predefined macro variables](#) on page 691.

## 14.9 Configuring variant booklist components

A bookmap can contain multiple instances of the same type of generated list. For **DITA2Go** to get the correct items into the correct list, you must configure separately any variants of a given booklist type.

*In this section:*

- §14.9.1 [Differentiating variant booklist components](#) on page 214
- §14.9.2 [Naming variant booklist components](#) on page 214
- §14.9.3 [Specifying properties of variant booklist components](#) on page 215
- §14.9.4 [Defining properties of items in variant booklist components](#) on page 216
- §14.9.5 [Mapping indexterms to variant indexes](#) on page 217

## 14.9.1 Differentiating variant booklist components

Suppose you want to produce two tables of contents for HTML output: an overview and a more detailed TOC showing more levels of headings. You can differentiate the two TOCs in DITA by assigning them different `outputclass` values; for example:

```
<frontmatter>
  <booklists>
    <toc outputclass="overview" />
    <toc outputclass="fulltoc" />
  . . .
```

To identify generated lists for which you want **DITA2Go** to produce two or more booklist variants, you must:

- distinguish variants by `@outputclass`
- assign each variant a name.

For example, to distinguish two lists of figures:

```
<frontmatter>
  <booklists>
    <figurelist outputclass="schematics" />
    <figurelist outputclass="assemblies" />
  . . .
```

To assign each variant a name, see §14.9.2 [Naming variant booklist components](#) on page 214.

## 14.9.2 Naming variant booklist components

To specify names for variant booklist components:

```
[BookLists]
; element/@outputclass = name
```

For example:

```
[BookLists]
figurelist/schematics = S_LOF
figurelist/assemblies = A_LOF
```

You do not have to assign names to booklist components for which you are not configuring variants. If a booklist component is not listed in `[BookLists]`, its name is the same as the element name.

The booklist components for which you can specify variants include:

```
abbrevlist
amendments
bibliolist
booklist
figurelist
glossarylist
indexlist
```

```
tablelist
trademarklist
```

In addition to these components, **DITA2Go** recognizes as a booklist component any element with the following properties (see §11.2 [Specifying properties of element types](#) on page 179):

```
Map
Ref
Topic
List
```

This automatically includes elements that are specialized from any of the recognized booklist types. See §B [Element type default properties](#) on page 825.

*TOC is different* Because TOC processing is very different, assigning a variant name directly does not work. However, for the TOC you can assign a variant name to the booklist element instead. For example:

```
[BookLists]
booklist/overview = OverviewTOC
```

*Index is different* In addition to specifying names and properties for indexlist variants and items, you must map indexterm outputclass values (assigned via PI) to variant indexes; see §14.9.5 [Mapping indexterms to variant indexes](#) on page 217.

### 14.9.3 Specifying properties of variant booklist components

In a general configuration file, create a section named for the booklist variant, with suffix BList. This is a variable-name, fixed-key section:

```
[VariantNameBList]
```

If you omit any properties, their default values will be those of the corresponding properties in effect for a normal list or index.

Properties of a booklist variant named in [BookLists] include:

Identifiers: [file name, suffix, reference ID](#)  
 Output formats: [for title, item, links, TOC entry](#)  
 Output configuration: [for index references](#)  
 Alphabetic aids: [for index navigation](#)  
 Title: [text of the title for the list](#)

*Identifiers: file  
name, suffix,  
reference ID*

To specify file name, suffix, and a reference ID for the booklist variant:

```
[VariantNameBList]
; FileName = Full name, no ext; if omitted, project base name + Suffix
FileName = D:\MyProject\Output\SchematicsLOF
; Suffix = Suffix for base name if no FileName provided, with defaults
; for some types, as listed in Table 14-1
Suffix = LOF
; RefID = ID to use for internal references such as "Rlof";
; default is R + variant name; for example:
RefID = Rs_lof
```

*Output formats:  
for title, item,  
links, TOC entry*

To specify output formats for booklist components:

```
[VariantNameBList]
; TitleFormat = block format to use for list title
TitleFormat = LOFTitle
; UseHeading = Yes (default, put a title on the index) or No
UseHeading = Yes
; ItemFormat = block format for items to include
ItemFormat = FigureListItem
```

```

; XrefFormat = inline format for list-item xrefs (not for indexlists)
XrefFormat = FigureListXref
; TOCFormat = block format to use for TOC entry for list
TOCFormat = LOFTOC

```

If the booklist variant includes items at different levels, the value of `ItemFormat` provides a default format for any levels not specified; see §14.9.4 [Defining properties of items in variant booklist components](#) on page 216.

*Output  
configuration: for  
index references*

To specify an output configuration for index references (indexlist variants only):

```

[IndexNameBList]
; CompactForm = Yes (default), page numbers (RTF) or icons (HTML);
; or No, full topic titles as index references
CompactForm = Yes
; RefFormat = inline format for index references (indexlists only)
RefFormat = IndexIconXref
; UseLeader = No (default) or Yes (RTF only) include a tab with
; a leader between index entry and first page number
UseLeader = No

```

See §14.8.4 [Configuring index references](#) on page 209.

*Alphabetic aids:  
for index  
navigation*

To include heading letters to introduce each alphabetic section of the index, and an alphabet across the top of each index page with links to the heading letters (indexlist variants only):

```

[IndexNameBList]
; UseLetters = Yes (default, insert heading letters) or No
UseLetters = Yes
; UseTopLetters = Yes (default for RTF; for HTML, default only if
; IndexNavType=HTML: top-of-page alphabet with links to heading
; letters) or No
UseTopLetters = Yes

```

See §14.8.5 [Including heading letters in the index](#) on page 210.

*Title: text of the  
title for the list*

To specify text for the title of the list or index: in a language configuration file, create a section named for the variant, with suffix `BLText`. This is a variable-name, fixed-key section where you specify the text string for the title of the list:

```

[VariantNameBLText]
; ListTitle = title to use if no navtitle in bookmap
ListTitle = Text of title

```

## 14.9.4 Defining properties of items in variant booklist components

For booklist variants other than `indexlist`: in a general configuration file, create a section named for the booklist variant, with suffix `BLItems`. This is a variable-name, variable-key section where you list the document formats of the items to be included in the list, and assign each a level number:

```

[VariantNameBLItems]
; DocFormatName = level in list, default 1
SomeTitle = 1

```

For example:

```

[OverviewBLItems]
ChapterTitle = 1
Heading1SC = 2
Heading1C = 2

```

The formats you list here are *not* the formats to be used to render the items as part of the generated list. You specify those formats in `[VariantNameBList]`; see §14.9.3 [Specifying properties of variant booklist components](#) on page 215. Instead, this section

enumerates the formats used in your document to render the content of the items. In other words, **DITA2Go** uses the format names you list here as selectors; see §7.1 [Understanding the purpose of output formats](#) on page 109.

For example, a `<figurelist>` would include figure titles as list items; so you would include in `[VariantNameBLItems]` the format used to render the `<title>` elements of `<fig>`s where they appear with images in output:

```
[AssembliesBLItems]
FigureTitle = 1
```

**Note:** Do not use `[VariantNameBLItems]` for `indexlist` variants; selectors for `indexterms` are derived from `outputclass` values rather than format names. See §14.9.5 [Mapping indexterms to variant indexes](#) on page 217.

*Formats for list items other than in indexes*

For booklist variants that contain items at more than one level, such as TOC and index variants: in a general configuration file, create a section named for the booklist variant, with suffix `BLForms`. This is a variable-name, variable-key section where you list the levels, and associate an output format with each level:

```
[VariantNameBLForms]
; level number = item format
```

The formats you assign to levels here *are* the formats to be used to render the items as part of the generated list. For example:

```
[OverviewBLForms]
1 = Heading1TOC
2 = TOCItem
```

Typically, items at each successive level would be assigned a format that indents the content relative to the previous level.

*Formats for list items in indexes*

You do not need to assign formats to index entries, except to override the default formats; see §14.8.2 [Overriding formats for index entries and references](#) on page 207. For unusual cases, you can assign formats to levels for the references from index entries to content:

```
[VariantNameBLRefForms]
; level number = reference format
```

For example:

```
[RTF_IXBLRefForms]
1 = MyIndexRef1
2 = MyIndexRef2
3 = MyIndexRef3
```

See also §14.9.5 [Mapping indexterms to variant indexes](#) on page 217.

## 14.9.5 Mapping indexterms to variant indexes

Each `indexterm` you want to include in a variant `indexlist` must be immediately preceded in your DITA source by a PI of the form:

```
<?dtall outputclass="variantname" ?>
```

This is because the `outputclass` attribute is not allowed on `indexterm`. See §38.1.1 [Understanding DITA2Go PI marker syntax](#) on page 717.

For `indexlist` variants, in addition to names and properties, you need settings to accomplish the following:

[Map outputclass values to variants](#)

[List outputclass values for each variant.](#)



*Map outputclass values to variants*

To map outputclass PI values to the indexlist variants named in [BookLists] (see §14.9.2 [Naming variant booklist components](#) on page 214):

```
[IndexClasses]
; indexterm outputclass PI = name(s) of indexlist variant(s)
```

In addition to names of indexlist variants, you can use as values two predefined flags:

IDX	Identifies the normal index
NoIDX	Omits from all indexes the content of any indexterms with the specified outputclass value

An indexterm with an outputclass PI value mapped in [IndexClasses] is included only in the indexlist variants to which it is mapped; if the IDX flag is in the list of variants, that includes the normal index. You need the NoIDX flag only for those indexterms you do not want in any index.

You can map an outputclass value to more than one indexlist variant. For example:

```
[IndexClasses]
Subject = TestIX IDX
Test = TestIX
Skip = NoIDX
```

An indexterm is included *only* in the normal index if it has any of the following:

- an outputclass PI value mapped only to IDX
- an outputclass PI value that is not mapped
- no outputclass PI.

*List outputclass values for each variant*

To specify the outputclass PI values of indexterms to include in each indexlist variant:

```
[IndexLists]
; indexlist name = outputclass PI values to include
TestIX = Subject Test
```

The normal index includes the content of all indexterms with the following characteristics:

- no outputclass PI
- an outputclass PI value not mapped in [IndexClasses]
- an outputclass PI value mapped to IDX in [IndexClasses].

To omit an indexterm from all indexes, give the indexterm element an outputclass PI, and map the value to NoIDX in [IndexClasses].



# 15 Converting to print RTF

---

This section shows you how to specify options for converting to Microsoft Word. The RTF produced for Word can also be viewed in OpenOffice and StarOffice. Topics include:

- §15.1 [Setting up a print RTF project](#) on page 219
- §15.2 [Adjusting output for different versions of Word](#) on page 224
- §15.3 [Converting paragraph and character formats](#) on page 225
- §15.4 [Modifying text appearance](#) on page 227
- §15.5 [Converting cross references and hypertext links](#) on page 229
- §15.6 [Converting tables to print RTF](#) on page 232
- §15.7 [Managing graphics for print RTF](#) on page 234
- §15.8 [Including RTF code for Word output](#) on page 238
- §15.9 [Turning on revision tracking in Word](#) on page 239
- §15.10 [Managing Word output after conversion](#) on page 239
- §15.11 [Converting to OpenOffice or StarOffice](#) on page 241

If you are creating WinHelp, see:

- §16 [Producing on-line Help](#) on page 243
- §17 [Generating WinHelp](#) on page 281

You must use a separate project directory and separate configuration files for WinHelp; Word and WinHelp RTF files are not compatible.

## 15.1 Setting up a print RTF project

To add or change any of the options described in this section, edit configuration file `_d2rtf.ini`, located in the project directory.

*In this section:*

- §15.1.1 [Specifying output file extension](#) on page 219
- §15.1.2 [Specifying the default output language and code page](#) on page 220
- §15.1.3 [Constraining the number of bookmarks in Word](#) on page 221
- §15.1.4 [Including or excluding contents and index for RTF output](#) on page 221
- §15.1.5 [Producing PDF automatically via Word](#) on page 222
- §15.1.6 [Launching Word from the DITA2Go Project Manager](#) on page 223
- §15.1.7 [Importing a Word template](#) on page 223

### 15.1.1 Specifying output file extension

By default, **DITA2Go** 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 **DITA2Go** produces must be loaded in Word and then saved as the desired output; see §15.10.1 [Supporting more than one version of Word](#) on page 239.

If you are converting to WordPerfect, also specify:

```
[WordOptions]
; WordPerfect = No (default) or Yes to override all features
; WP does not tolerate
WordPerfect = Yes
```

### 15.1.2 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 DITA document.

You can use the following decimal Unicode values:

US English	1033 (default)
UK English	2057
Oz English	3081
German	1031

**DITA2Go** 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
```

### 15.1.3 Constraining the number of bookmarks in Word

By default, **DITA2Go** 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
```

Omitting unneeded related links can drastically reduce the number of bookmarks. For example, to produce the **DITA2Go User's Guide**, all but the child links had to go:

```
[RelatedLinks]
AppendLinksToTopics = Yes
GeneratePrevNext = No
GenerateParentChild = Yes
GenerateSiblings = No
UsePrevNext = No
UseParent = No
UseChildren = No
UseSiblings = No
UseAncestors = No
UseDescendants = No
UseCousins = No
UseFriends = No
UseRelatedDivider = No
```

See §13.3 [Appending links to topics](#) on page 190.

### 15.1.4 Including or excluding contents and index for RTF output

**DITA2Go** generates a table of contents for Word, by default, from map information. The table of contents precedes the first topic; see §14.3 [Generating a table of contents](#) on page 198.

To omit the table of contents:

```
[Contents]
; GenerateTOC = Yes (default) or No (omit TOC)
GenerateTOC = No
```

If your DITA source has a bookmap that includes an `<indexlist>` element, by default **DITA2Go** generates Word index entries from any index terms in your document. See §14.8 [Producing an index](#) on page 206.

To omit the index:

```
[Index]
; GenerateIDX = Yes (default, generate index) or No.
GenerateIDX = No
```

When GenerateIDX=Yes, provided [WordOptions]Index=Standard, you can have Word generate an index after conversion. To include {xe} fields for index terms in RTF output, see §15.10.2 [Including index terms in Word](#) on page 240.

For RTF output, **DITA2Go** does not produce a formatted index; instead, you must update fields in Word for this step. The result will be an index with page-number references, but without live links to content.

See also:

§15.1.5 [Producing PDF automatically via Word](#) on page 222

## 15.1.5 Producing PDF automatically via Word

Starting with Word 2007, Microsoft Word can output PDF with no added tools. This means that you can run your entire **DITA2Go** conversion, from ditamap to PDF, as one step, using a Word macro to get from RTF to PDF. For earlier versions of Word, you can use Adobe PDF as your printer if you have Adobe Acrobat (not just Reader). Or, there are dozens of other PDF makers, many of them free.

**Note:** A defect in the 64-bit version of Office 2010 might cause Word to crash when you try to create a PDF from your **DITA2Go** Word output. The remedy is to use the 32-bit version of Word 2010, even if you are on a 64-bit Windows 7 system.

What you need is a Word macro that will do the following upon loading an RTF file:

1. Update fields
2. Generate the index, if you included index terms (see §15.10.2 [Including index terms in Word](#) on page 240)
3. Save as .doc
4. Generate a PDF file, with active links (*except* from index entries).

PDF macro for  
Word 2007

The following Word macro is designed for Word 2007:

```
Sub LoadD2gRtfFile()
'
' LoadD2Grtf2007 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
nameStr = Replace(ActiveDocument.FullName, ".doc", ".pdf")
ActiveDocument.ExportAsFixedFormat OutputFileName:=nameStr, _
    ExportFormat:=wdExportFormatPDF, _
    OptimizeFor:=wdExportOptimizeForPrint, _
    Range:=wdExportAllDocument, From:=1, To:=1, _
    Item:=wdExportDocumentContent, KeepIRM:=True, _
    CreateBookmarks:=wdExportCreateNoBookmarks, _
    DocStructureTags:=True, BitmapMissingFonts:=True, _
    UseISO19005_1:=False
Selection.StartOf
End Sub
```

This particular macro works in Word 2007, provided you uncheck the “rely on system fonts only” option in Adobe PDF printing preferences. You might have to modify the macro for other versions of Word. Try to run it from within Word after you load the RTF and save it as .doc (or .docx). See if it works then. Also, make sure the name of the macro is correct. Depending on which version of Word you use, the resulting PDF file will have active links, except for index entries.

*Settings required  
for PDF  
production*

For automatic PDF production you must also include the following settings in your **DITA2Go** project configuration file:

```
[Setup]
; To get the correct extension after Word saves as .doc:
FileSuffix=.doc

[WordOptions]
; To get active links, you must target Word 2007 or a later version:
Word2007 = Yes
; To run the macro from the Project Manager:
ViewOutputCommand = path/to/winword.exe /mLoadD2gRtfFile
```

*PDF from the  
Project Manager*

When you click **View Output** on the **DITA2Go** Project Manager **Run Project** tab, the Project Manager loads your RTF output file in Word. The macro runs, producing a PDF with the same base name as your Word RTF file, and extension .pdf. See §15.1.6 [Launching Word from the DITA2Go Project Manager](#) on page 223.

*PDF from the  
command line*

To produce PDF via Word from the command line:

```
dc1 -f rtf -o .doc ..\mymap.bookmap "path/to/winword.exe"
/mLoadD2gRtfFile mymap.doc
```

This command must be all on one line, even if it does not look that way here. See §2.7 [Converting documents from the command line](#) on page 46

### 15.1.6 Launching Word from the DITA2Go Project Manager

When you use the **DITA2Go** Project Manager to run a conversion to Word, you can view the output immediately with the **View Output** button on the **Run Project** tab. You can even have Word run a macro at the same time, with the Word /m switch. For example, to launch Word and then run a macro that will *Select all/Update fields/Deselect*:

```
[WordOptions]
; ViewOutputCommand = path\to\viewer.bat, default none
ViewOutputCommand=C:\office2k\Office\WINWORD.EXE /mUpdateSaveDoc
```

You can specify an absolute path or a path relative to the wrap directory. See §1.3.7 [Establish system-wide configuration settings](#) on page 33.

### 15.1.7 Importing a Word template

To format RTF output, you must either use the formatting options provided by **DITA2Go**, or supply your own Word template.

To import a Word template:

```
[WordOptions]
; Template is one of the two possible sources of formatting info, and
; must be present if [Templates]Formats is not used.
Template = RequiredName.dot
```

A better alternative to importing a Word template is to specify the Word styles you want in terms of **DITA2Go** output formats; see §7 [Configuring output formats](#) on page 109. Use a Word template only as a last resort.

## 15.2 Adjusting output for different versions of Word

Microsoft makes significant changes to the underpinnings of every new version of Word. RTF output from **DITA2Go** that looks fine in one version of Word might not look quite right in another version.

Consider which version(s) of Word will be used to view your **DITA2Go** 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, **DITA2Go** produces RTF output tuned for Word 2003.

*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 15-1](#) lists the main differences in the RTF code that **DITA2Go** generates, based on the value specified for Word8.

**Table 15-1 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	<a href="#">15.5</a>
Table straddles	Column only	Row and column	
Table cell vertical alignment	No	Yes	
Graphics scale units	twips	himetric (Word 8, 9, and 10 only)	<a href="#">15.7.4</a>
Image field name	IMPORT	INCLUDEPICTURE	

*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 [§15.7.4 Preserving graphics scale in Word](#) on page 236.

Differences in how Word interprets RTF code are particularly noticeable in the following document features:

- Graphics      Scaling units vary among Word versions; see [§15.7.4 Preserving graphics scale in Word](#) on page 236.
- Tables        Straddled rows and rotated text are supported only by some Word versions; see [§15.6 Converting tables to print RTF](#) on page 232.

Links	Cross references and hypertext links can be made active and updatable in later versions of Word; see §15.5 <a href="#">Converting cross references and hypertext links</a> on page 229.
Revision tracking	Revision tracking can be turned on automatically in some versions of Word; see §15.9 <a href="#">Turning on revision tracking in Word</a> on page 239.
Multiple versions of Word	If your RTF output files will be viewed in multiple versions of Word, see §15.10.1 <a href="#">Supporting more than one version of Word</a> on page 239.

## 15.3 Converting paragraph and character formats

Sometimes differences in available formats in Word can make precise conversions tricky. You might have to experiment with different settings to get the appearance you want.

*In this section:*

§15.3.1 [Mapping paragraph formats to RTF styles](#) on page 225

§15.3.2 [Merging paragraph formats](#) on page 226

§15.3.3 [Converting autonumbered formats](#) on page 226

§15.3.4 [Converting bulleted formats](#) on page 226

§15.3.5 [Converting character formats](#) on page 227

### 15.3.1 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* 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 §15.3.2 [Merging paragraph formats](#) on page 226.

*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* DITA2Go ignores any assignment in the [Styles] section that has no entry to the right of the equals sign.



### 15.3.2 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
```

**DITA2Go** ignores any assignment in the [StyleReplacements] section that has no entry to the right of the equals sign.

### 15.3.3 Converting autonumbered formats

To convert **DITA2Go** 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, **DITA2Go** adds numbering prefixes to text paragraphs as SEQ fields, according to numbering properties you assign to paragraph formats (see §8.5 [Configuring output numbering properties](#) on page 146) and format definitions you specify in a format configuration file (see §7.5 [Understanding text output formats](#) on page 119).

When WriteAnums=No, the value of SeqAnums is ignored, and **DITA2Go**-supplied numbering is omitted in favor of whatever Word stylesheet you supply. This is not a recommended option, because Word style-based numbering is very problematic.

By default, the tabs included in autonumber formats (see §8.5 [Configuring output numbering properties](#) on page 146), are 1/4" or 18pt wide. To specify a different width, in points:

```
[WordOptions]
; AnumTabWidth = width in points for anum/xref tab spacing, default
; 1/4", for the tabs specified in [*Formats]; used only for drxml
AnumTabWidth = 18
```

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*.

### 15.3.4 Converting bulleted formats

Bulleted lists in DITA should convert to their RTF equivalents without a problem, except in the following circumstance:

- You are converting to Word 97.
- The 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
```

### 15.3.5 Converting character formats

**DITA2Go** can output 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 **DITA2Go** 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 **DITA2Go** puts out overrides: there is no choice, in Word.

## 15.4 Modifying text appearance

*In this section:*

- §15.4.1 [Adjusting line spacing](#) on page 227
- §15.4.2 [Specifying a style for quotes](#) on page 227
- §15.4.3 [Hiding content in Word](#) on page 228
- §15.4.4 [Omitting content from RTF output](#) on page 228
- §15.4.5 [Replacing content in RTF output](#) on page 228

### 15.4.1 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
```

### 15.4.2 Specifying a style for quotes

Converting to Word for print output, you might choose to use “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
```

### 15.4.3 Hiding content in Word

To convert content so it becomes hidden text in Word, set a unique value of `@outputclass` for the material in DITA XML, and assign the following property to the resulting 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 §15.4.4 [Omitting content from RTF output](#) on page 228.

### 15.4.4 Omitting content from RTF output

To prevent text from appearing in RTF output, you can do the following:

1. Use a special value of `@outputclass` (for example, `onlineonly`)
1. Use a special value of `@outputclass` (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
```

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
```

### 15.4.5 Replacing content in RTF output

*This method is deprecated in favor of the `CodeReplace` property described in §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.*

You can direct **DITA2Go** 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:

§37.3.5.2 [Inserting code with the `CodeStore` property](#) on page 693.

## 15.5 Converting cross references and hypertext links

*In this section:*

§15.5.1 [Converting cross references to Word](#) on page 229

§15.5.2 [Converting hypertext links to Word](#) on page 231

§15.5.3 [Locking hypertext links to allow revision tracking](#) on page 231

### 15.5.1 Converting cross references to Word

Cross references can become active links in Word, and they can be updated in Word.

*In this section:*

§15.5.1.1 [Understanding how DITA2Go converts cross references](#) on page 229

§15.5.1.2 [Making cross references active and updatable](#) on page 229

§15.5.1.3 [Weighing cross-reference behavior trade-offs](#) on page 230

§15.5.1.4 [Omitting cross references from RTF output](#) on page 231

#### 15.5.1.1 Understanding how DITA2Go converts cross references

When you specify Word output, cross references in your DITA document become Word bookmarks and bookmark-references. 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, **DITA2Go** also wraps cross references as Word hypertext links.

#### 15.5.1.2 Making cross references active and updatable

For Word output, by default **DITA2Go** 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 ( <b>Ctrl</b> -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 15-2 summarizes the effects of these settings.

**Table 15-2 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`.

*Referencing page numbers in different files*

Word does not support page references for cross-reference destinations in a different file, so **DITA2Go** 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.

### 15.5.1.3 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.

### 15.5.1.4 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 §15.1.3 [Constraining the number of bookmarks in Word](#) on page 221.

## 15.5.2 Converting hypertext links to Word

By default, **DITA2Go** creates active links in Word from hypertext links:

```
[WordOptions]
; UseHyperlinks = Yes (default) or No (ignore all hyperlinks)
UseHyperlinks=Yes
```

## 15.5.3 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 §15.9 [Turning on revision tracking in Word](#) on page 239.

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.

## 15.5.4 Enabling interfile cross references and hypertext links

§15.5.4.1 [Making page numbers in interfile links updatable](#) on page 231

§15.5.4.2 [Mapping file names, extension, or location](#) on page 231

### 15.5.4.1 Making page numbers in interfile links updatable

By default, **DITA2Go** 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 §15.5.1.2 [Making cross references active and updatable](#) on page 229), 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).

### 15.5.4.2 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 §15.1.1 [Specifying output file extension](#) on page 219)
- the same base names as the corresponding DITA 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 §15.1.1 [Specifying output file extension](#) on page 219).
- You specified names for output files that differ from names of the corresponding input files (see §45.4 [Specifying output file paths and names](#) on page 812).
- 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-DITA2Go 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**.

## 15.6 Converting tables to print RTF

In Word, tables are not objects. They are just paragraphs with border properties that make them look like rows.

Before you attempt the fine tuning described in this section:

- To specify output formats and general options for DITA table types, see §6.9 [Specifying formats and options for tables](#) on page 103.
- To map table elements to output formats using attributes, see §6.3.2 [Mapping table outputclass attributes to formats](#) on page 90.
- To define properties for each table format, see §7.7 [Configuring table output formats](#) on page 129.

You can also use configuration settings or PI markers to control some table characteristics for RTF output:

[Table and column widths](#)

Table title position

Cell properties

Straddled columns

Other characteristics might be more problematic:

Straddled rows

Tables with too many columns

Table and column widths

Define default widths for tables and table columns in a table-format configuration file; see §7.7 [Configuring table output formats](#) on page 129. To specify a different table width for an individual table generated from DITA `<simplatable>`, `<choicetable>`, or `<properties>` element, insert a **SimpleTableWidth** PI marker just before or at the beginning of the element. For example:

```
<?dtrtf SimpleTableWidth="4.5in" ?>
```

DITA definition lists and parameter lists are best rendered as tables in RTF output. These element types do not allow width settings in DITA XML. However, you can specify either relative or absolute column widths for RTF output, with a **SimpleTableRelCol** PI marker placed just before or at the beginning of the element. For example, for relative column widths:

```
<?dtrtf SimpleTableRelCol="2* 4*" ?>
```

This simply acts exactly like DITA `@relcolwidth` for the tables generated from those lists. For absolute column widths, you can use any units allowed in DITA; for example:

```
<?dtrtf SimpleTableRelCol="1in 36pt 5pc" ?>
```

You can combine both **SimpleTableWidth** and **SimpleTableRelCol** values in a single PI marker, separated by a space; see §38.1.1 [Understanding DITA2Go PI marker syntax](#) on page 717.

Table title position

To reposition table titles:

```
[Tables]
; TableTitles = 0 to leave alone, 1 to put at top, 2 to put at bottom
TableTitles=0
```

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, and override values of the same properties specified in a table-format configuration file; see §7.7.4

[Configuring cell format properties](#) on page 132.

`TableRules` determines whether the border values set in a table-format configuration file take effect. If `TableRules` has any value except `Cell`, **DITA2Go** does not write borders. If `TableFill=None` or `ColorOnly`, **DITA2Go** ignores the table-format configuration settings for shading.

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 **DITA2Go** 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 §17.5.2 [Adjusting table appearance](#) on page 291.

*Straddled rows* Word 8/97 and later versions can handle straddled rows in tables; Word 7/95 cannot.

*Tables with too many columns* 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.

## 15.7 Managing graphics for print RTF

*In this section:*

- §15.7.1 [Understanding graphics requirements for Word](#) on page 234
- §15.7.2 [Understanding where to locate graphics](#) on page 235
- §15.7.3 [Limiting bitmap resolution and color depth](#) on page 235
- §15.7.4 [Preserving graphics scale in Word](#) on page 236
- §15.7.5 [Accommodating graphics in multiple versions of Word](#) on page 237
- §15.7.6 [Including file names of referenced graphics in Word](#) on page 237
- §15.7.7 [Linking instead of embedding referenced graphics](#) on page 237
- §15.7.8 [Embedding graphics in converted RTF files](#) on page 238
- §15.7.9 [Updating fields in Word to show linked graphics](#) on page 238

*See also:*

- §40 [Working with graphics](#) on page 745

### 15.7.1 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 Word allows images to be scaled only if the images are embedded as WMFs. Only WMF or BMP images can be embedded as WMFs in Word. **DITA2Go** 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 DITA document (or substituted for the images referenced by your DITA 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 §15.7.7 [Linking instead of embedding referenced graphics](#) on page 237

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 **DITA2Go** has to take each such WMF graphic apart and adjust many settings before embedding the image.

If your DITA document references graphics that are neither BMP nor WMF, you must provide BMP or WMF replacements for the graphics files.



For referenced graphics in other formats, **DITA2Go** 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.

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.

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.

*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 §40.2.2.1 [Substituting graphics files for RTF](#) on page 748.

*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 §4.4 [Processing graphics](#) on page 77.

## 15.7.2 Understanding where to locate graphics

For RTF output, graphics files should end up in the same directory as the RTF files; otherwise, if you move the project you are very likely to break links.

If the graphics are BMP or WMF, **DITA2Go** can embed them directly in RTF. If other formats, **DITA2Go** writes a field that causes Word to embed them on loading the RTF. In the latter case, you must select the correct images to copy, and direct **DITA2Go** to copy them; see §44.7 [Placing graphics files for distribution](#) on page 796.

Once you resave an RTF file in Word as a `.doc` or `.docx` file, there is no further need for the graphics files; the path is used only during conversion. Word always embeds images, even if you specify referencing in the field code.

## 15.7.3 Limiting bitmap resolution and color depth

Graphics destined for print look best at a resolution of 300 to 1,200 DPI. However, you cannot 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 **DITA2Go** has to

Save Word files  
with 24-bit BMP  
images as .doc

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.

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, **DITA2Go** cannot legally use this method, thanks to the Digital Millennium Copyright Act (DMCA).

## 15.7.4 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, **DITA2Go** 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 15-3** shows what happens to the apparent size of an embedded image viewed in Word, depending on which version of Word you specify as the **DITA2Go** 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 §15.7.7 [Linking instead of embedding referenced graphics](#) on page 237.

**Table 15-3 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.

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.

### 15.7.5 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:*

§15.2 [Adjusting output for different versions of Word](#) on page 224.

§15.7.7 [Linking instead of embedding referenced graphics](#) on page 237.

### 15.7.6 Including file names of referenced graphics in Word

*Name and graphic* To include the original file names of 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, **DITA2Go** 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 §40.2.2.3 [Excluding graphics from RTF output](#) on page 751.

### 15.7.7 Linking instead of embedding referenced graphics

You can use **DITA2Go** 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 §15.7.1 [Understanding graphics requirements for Word](#) on page 234.

To create RTF files without embedding referenced graphics, so that Word can link to the graphics files instead, make sure that **DITA2Go** can neither find nor generate WMF or BMP versions of those graphics. Make sure no referenced graphics are in the project directory. Then **DITA2Go** will create INCLUDEPICTURE references to the graphics instead of embedding them.

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:*

§15.2 [Adjusting output for different versions of Word](#) on page 224

§15.7.6 [Including file names of referenced graphics in Word](#) on page 237

§15.7.8 [Embedding graphics in converted RTF files](#) on page 238

### 15.7.8 Embedding graphics in converted RTF files

If your conversion project is set up so that **DITA2Go** 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:*

§15.7.6 [Including file names of referenced graphics in Word](#) on page 237

§15.7.7 [Linking instead of embedding referenced graphics](#) on page 237

### 15.7.9 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.

## 15.8 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 **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** conversion, you should see:

```
{\\field {\\*\\fldinst INCLUDEPICTURE "MyLogo.bmp" \\d \\x \\y
\\* MERGEFORMAT }\\fldrslt }}
```

Line breaks are acceptable in RTF code.

## 15.9 Turning on revision tracking in Word

To turn on revision tracking in Word for **DITA2Go** 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 §15.5.1 [Converting cross references to Word](#) on page 229), you get change bars in Word for every cross reference; and if you also specify live hypertext links (see §15.5.2 [Converting hypertext links to Word](#) on page 231), 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.

## 15.10 Managing Word output after conversion

*In this section:*

§15.10.1 [Supporting more than one version of Word](#) on page 239

§15.10.2 [Including index terms in Word](#) on page 240

§15.10.3 [Producing ASCII text from a converted Word document](#) on page 241

§15.10.4 [Checking print RTF output files for DITA2Go version](#) on page 241

### 15.10.1 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 **DITA2Go**:

1. Load each generated RTF file into the version of Word for which **DITA2Go** produced it; see §15.2 [Adjusting output for different versions of Word](#) on page 224).

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; for example, 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.

Also see §15.1.5 [Producing PDF automatically via Word](#) on page 222.

## 15.10.2 Including index terms in Word

To have **DITA2Go** include index terms in RTF output:

```
[WordOptions]
; Index = Standard (Word index markers), or None
Index=Standard
```

When Index=Standard, **DITA2Go** creates a Word {xe} field for each <indexterm>. To generate an index from these fields in Word, see §15.1.4 [Including or excluding contents and index for RTF output](#) on page 221.

**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. However, when you specify Index=Standard you can direct **DITA2Go** to refrain from converting see-also index entries to Word {xe} fields:

```
[WordOptions]
; NoSeeAlso = No (default, keep See Also markers)
; or Yes (remove them)
NoSeeAlso=Yes
```

When NoSeeAlso=No, see-also index entries are converted to {xe} fields in RTF output files.

When NoSeeAlso=Yes, see-also index entries are not converted to {xe} fields in RTF output.



See also:

§14.8 [Producing an index](#) on page 206

§15.1.4 [Including or excluding contents and index for RTF output](#) on page 221

### 15.10.3 Producing ASCII text from a converted Word document

If your reason for converting a document from DITA 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 §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

*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 **DITA2Go** uses to emulate dynamic cross references to pages. To omit these numbers:

```
[WordOptions]
ExtXrefPages = No
```

See §15.5.4.1 [Making page numbers in interfile links updatable](#) on page 231.

### 15.10.4 Checking print RTF output files for DITA2Go version

If you recently installed a **DITA2Go** upgrade or beta version, after you run **DITA2Go**, 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 d194b r278b
```

The last two entries identify the build numbers of the **DITA2Go** `drxml.dll` and `dwrtf.dll` components that were used to create the RTF file. See §A.1.5 [Check your version of DITA2Go](#) on page 820.

## 15.11 Converting to OpenOffice or StarOffice

OpenOffice.org Writer and StarOffice can open the RTF files that **DITA2Go** converts from DITA 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.



# 16 Producing on-line Help

---

You can use **DITA2Go** 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:

- §16.1 [Weighing Help-system alternatives](#) on page 243
- §16.2 [Completing Help system construction](#) on page 247
- §16.3 [Producing contents and index for Help systems](#) on page 248
- §16.4 [Configuring contents entries for Help systems](#) on page 250
- §16.5 [Configuring index entries for Help systems](#) on page 251
- §16.6 [Providing related-topic links for Help systems](#) on page 258
- §16.7 [Jumping to secondary windows in Help systems](#) on page 262
- §16.8 [Creating pop-up topics for Help systems](#) on page 263
- §16.9 [Including expandable sections in Help topics](#) on page 264
- §16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277
- §16.11 [Setting up a dynamic modular Help system](#) on page 280

For strategies and configuration settings that are specific to a particular Help system, see the following:

- §17 [Generating WinHelp](#) on page 281
- §18 [Generating Microsoft HTML Help](#) on page 313
- §19 [Generating OmniHelp](#) on page 353
- §20 [Generating JavaHelp or Oracle Help](#) on page 385
- §21 [Generating Eclipse Help](#) on page 413

## 16.1 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 §22.13 [Using framesets](#) on page 443). 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:*

- §16.1.1 [Considering Help-system features](#) on page 244
- §16.1.2 [Understanding the effects of mid-topic links](#) on page 244
- §16.1.3 [Evaluating Microsoft Windows Help \(WinHelp\)](#) on page 244
- §16.1.4 [Evaluating Microsoft HTML Help](#) on page 245
- §16.1.5 [Evaluating WebHelp](#) on page 245
- §16.1.6 [Evaluating OmniHelp](#) on page 245

§16.1.7 [Evaluating JavaHelp and Oracle Help for Java](#) on page 246

§16.1.8 [Evaluating Eclipse Help](#) on page 246

### 16.1.1 Considering Help-system features

For HTML-based Help systems, **DITA2Go** 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 **DITA2Go** to produce WinHelp 2000. You generate WinHelp, then add the eHelp data to the WinHelp project file before you compile; see §17.2.9 [Integrating WinHelp from RoboHelp](#) on page 285. Also, you can generate HTML Help as a precursor to using RoboHelp to generate WebHelp; see §16.1.5 [Evaluating WebHelp](#) on page 245.

If you need something similar to WebHelp or Web Works Help, try generating OmniHelp; see §16.1.6 [Evaluating OmniHelp](#) on page 245. Otherwise you would have to roll your own, which you could do with **DITA2Go** frameset support; see §22.13 [Using framesets](#) on page 443. 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 **DITA2Go** 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 §16.5.2 [Preparing index entries for Microsoft Help Viewer](#) on page 252.

To produce electronic books and content for mobile devices, see §22.1 [Deciding which type of output to produce](#) on page 430.

### 16.1.2 Understanding the effects of mid-topic links

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.

### 16.1.3 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 §17 [Generating WinHelp](#) on page 281.

### 16.1.4 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.

### 16.1.5 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 **DITA2Go**, 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 §16.5.7 [Specifying index link destinations for HTML-based Help](#) on page 255.

### 16.1.6 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://www.dita2go.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 §19 [Generating OmniHelp](#) on page 353.

### 16.1.7 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. **DITA2Go** writes Oracle Help for Java files in Oracle Help preferred format, rather than just JavaHelp format. You can use most of the features **DITA2Go** 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 §20 [Generating JavaHelp or Oracle Help](#) on page 385.

### 16.1.8 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.

## 16.2 Completing Help system construction

### *In this section:*

§16.2.1 [Specifying additional processing after conversion](#) on page 247

§16.2.2 [Compiling and distributing Help systems](#) on page 247

§16.2.3 [Launching a Help viewer from the Project Manager](#) on page 248

### 16.2.1 Specifying additional processing after conversion

When you set up a new Help project, **DITA2Go** includes a few postprocessing settings in your newly created configuration file.

Some Help systems require running additional programs after files are converted from DITA: 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 **DITA2Go** project to generate one of these Help systems, you can include an option to have **DITA2Go** run the additional program after conversion.

*WinHelp, HTML  
Help*

For a WinHelp or HTML Help project, add the following settings to your configuration file:

```
[Automation]
CompileHelp = Yes
WrapAndShip = Yes
```

*JavaHelp, Oracle  
Help*

For JavaHelp or Oracle Help, include the following setting in your new configuration file:

```
[Automation]
WrapAndShip = Yes
```

*OmniHelp,  
Eclipse Help*

For OmniHelp or Eclipse Help, include the following setting:

```
[Automation]
WrapAndShip = Yes
```

If `CompileHelp=Yes` or `WrapAndShip=Yes`, after generating output files **DITA2Go** 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 `_wrap`, change the value of `WrapPath` in your configuration file. See §16.2.2 [Compiling and distributing Help systems](#) on page 247.

### 16.2.2 Compiling and distributing Help systems

After generating Help output files, **DITA2Go** 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 §44 [Producing deliverable results](#) on page 787.

*See also:*

§17.2.10 [Compiling a WinHelp project](#) on page 285

§18.13 [Compiling and testing HTML Help](#) on page 346

§19.13 [Assembling OmniHelp files for viewing](#) on page 380

§20.3.6 [Creating a directory structure for JavaHelp / Oracle Help](#) on page 389

§21.8 [Packaging Eclipse Help files](#) on page 427

### 16.2.3 Launching a Help viewer from the Project Manager

When you use the **DITA2Go** Project Manager to run a conversion, you can view the output immediately with the **View Output** button on the **Run Project** tab, provided both of the following are true:

- the key output file for the Help system is located in the wrap directory
- if a command is needed to launch the viewer, you have specified that command.

For OmniHelp, HTML Help, and WinHelp, you do not have to supply a command to start the viewer at the default topic, as long as the name of the key file (for OmniHelp) or the compiled system is located in the wrap directory. This is equivalent to specifying the default value (blank):

```
[HelpOptions] or [OmniHelpOptions] or [MSHtmlHelpOptions]
; ViewOutputCommand = path\to\viewer, default none
ViewOutputCommand =
```

For JavaHelp, Oracle Help, and Eclipse Help systems you will need to provide a command the Project Manager can launch to run the viewer.

```
[JavaHelpOptions] or [OracleHelpOptions] or [EclipseHelpOptions]
ViewOutputCommand = path\to\viewer
```

You can specify an absolute path or a path relative to the wrap directory. For example:

```
[JavaHelpOptions]
ViewOutputCommand = java -jar D:\JH2\demos\bin\hsvviewer.jar -helpset
```

For Oracle Help, the command can be quite long, and must be all on the same line, even if it does not look that way here:

```
[OracleHelpOptions]
ViewOutputCommand = java -classpath "d:\ohj\ohj427\help4.jar;
d:\ohj\ohj427\help4-demo.jar;d:\ohj\ohj427\ohj-jewt.jar;
d:\ohj\ohj427\;d:\ohj\ohj427\oracle_ice.jar"
oracle.help.demo.ChoiceDemo
```

For Eclipse Help, you are on your own.

See §1.3.7 [Establish system-wide configuration settings](#) on page 33.

## 16.3 Producing contents and index for Help systems

By default, **DITA2Go** generates both contents and index for Help systems:

- Contents entries are based on topic headings in your DITA document.
- Index entries are produced from DITA index entries.

However, you can choose to exclude contents or index or both.

*In this section:*

§16.3.1 [Modifying contents or index production for HTML-based Help](#) on page 249

§16.3.2 [Modifying contents or index production for WinHelp](#) on page 249



### 16.3.1 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.

See also:

§16.4 [Configuring contents entries for Help systems](#) on page 250

§16.5 [Configuring index entries for Help systems](#) on page 251

§18.9.1 [Choosing how to generate HTML Help contents and index](#) on page 335

§19.6 [Choosing navigation features for OmniHelp](#) on page 367

§20.4.4 [Locating JavaHelp or Oracle Help contents and index files](#) on page 397

§21.4.1 [Choosing contents and index methods for Eclipse Help](#) on page 420

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 or None
```

RefFileType values have the following effects:

Full	<b>DITA2Go</b> creates final contents and index files directly, depending on the value of ListType.
None	No 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.

For HTML Help, you can specify one additional value for RefFileType. See §18.9.1 [Choosing how to generate HTML Help contents and index](#) on page 335.

For OmniHelp, whether contents and index are displayed (as opposed to generated) is determined by another setting; see §19.6 [Choosing navigation features for OmniHelp](#) on page 367

### 16.3.2 Modifying contents or index production for WinHelp

*Contents* To specify how (and whether) contents entries are generated for WinHelp:

```
[HelpContents]
; CntType = None or Full (single file)
```

CntType values have the following effects:

Full	<b>DITA2Go</b> creates the final contents file, <i>MyDoc.cnt</i> , directly.
None	No .bct, or .cnt files are produced.

*Index* To specify whether index entries are generated for WinHelp from DITA index elements:

```
[HelpOptions]
; Index = Help (make into K footnotes) or None (removed)
```

See also:

§17.11 [Configuring index entries for WinHelp](#) on page 305

§17.12 [Configuring contents for WinHelp](#) on page 306

## 16.4 Configuring contents entries for Help systems

In this section:

§16.4.1 [Setting contents levels for WinHelp](#) on page 250

§16.4.2 [Including contents entries in HTML-based Help](#) on page 250

§16.4.3 [Setting contents levels for HTML-based Help](#) on page 251

### 16.4.1 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 §17.12.2 [Specifying heading formats and levels for contents](#) on page 307.

Each entry in [HelpCntStyles] must correspond to an entry with property Contents in [HelpStyles]; for example:

```
[HelpStyles]
Heading 1=Topic Contents
```

See §17.7.2 [Assigning properties to formats for topics and hotspots](#) on page 295.

See also:

§17.12 [Configuring contents for WinHelp](#) on page 306

### 16.4.2 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 §27.3.1 [Designating split points](#) on page 526. 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.

*Include modified titles in the TOC*

To use titles as modified by PI markers or by macros:

```
[MShtmlHelpOptions] or [JavaHelpOptions] or [OmniHelpOptions]
; UseNavtitleMarkers = No (default, use literal topic titles,
```



```
; or Yes (use titles as modified by title marker and macros
UseNavtitleMarkers = Yes
```

When `UseNavtitleMarkers=Yes`, **DITA2Go** includes in the TOC the `<navtitle>` element, if present, or the result of any macro you have specified for the title. See §14.3.4 [Including navigation titles from maps in the TOC](#) on page 200.

When `UseNavtitleMarkers=No`, **DITA2Go** includes in the TOC the titles present in topics or in `<topichead>` elements, without modification. So, for example, if you use a macro to add a product name to all the titles, that name does not have to clutter the TOC.

See also:

§16.4.3 [Setting contents levels for HTML-based Help](#) on page 251

§18.9.5 [Configuring contents entries for HTML Help](#) on page 337

§19.7 [Configuring contents and index for OmniHelp](#) on page 367

§20.4.1 [Configuring contents entries for JavaHelp or Oracle Help](#) on page 395

### 16.4.3 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 §16.4.2 [Including contents entries in HTML-based Help](#) on page 250. 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 §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566).

## 16.5 Configuring index entries for Help systems

*In this section:*

§16.5.1 [Understanding how DITA2Go creates Help index entries](#) on page 252

§16.5.2 [Preparing index entries for Microsoft Help Viewer](#) on page 252

§16.5.3 [Limiting length of index entries for HTML Help or WinHelp](#) on page 252

§16.5.4 [Treating commas as potential index level separators](#) on page 252

§16.5.5 [Combining index levels for HTML-based Help](#) on page 253

§16.5.6 [Configuring See and See also entries for HTML-based Help](#) on page 254

§16.5.7 [Specifying index link destinations for HTML-based Help](#) on page 255

§16.5.8 [Customizing index sort order](#) on page 256

See also:

§17.11 [Configuring index entries for WinHelp](#) on page 305

§18.9.8 [Customizing contents and index for HTML Help](#) on page 338

§20.4.3 [Configuring index entries for JavaHelp or Oracle Help](#) on page 396

## 16.5.1 Understanding how DITA2Go creates Help index entries

**DITA2Go** processes DITA index elements to create the index for a Help system.

For HTML-based Help, **DITA2Go** 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 **DITA2Go** can do to customize it.

## 16.5.2 Preparing index entries for Microsoft Help Viewer

Microsoft Help Viewer 1.x requires extra meta elements for index terms, so converting DITA `<indexterm>` elements to HTML or XHTML is not enough.

To have **DITA2Go** 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
```

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" />
```

## 16.5.3 Limiting length of index entries for HTML Help or WinHelp

**DITA2Go** 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
```

*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. **DITA2Go** 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.

## 16.5.4 Treating commas as potential index level separators

By default, **DITA2Go** treats commas in DITA index terms as potential level separators for indexes. However, **DITA2Go** 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 **DITA2Go** 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`, **DITA2Go** 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`, **DITA2Go** 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 §16.5.5 [Combining index levels for HTML-based Help](#) on page 253 and §16.5.6.2 [Specifying level breaks for See and See also index entries](#) on page 254.

#### 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 §17.11.1 [Designating index level separators](#) on page 305.

## 16.5.5 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, **DITA2Go** 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 **DITA2Go** 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, **DITA2Go** 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`, **DITA2Go** breaks index entries at all unescaped colons.

## 16.5.6 Configuring See and See also entries for HTML-based Help

If you tell **DITA2Go** the words used to introduce redirect references in your DITA index entries, **DITA2Go** 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.

You can direct **DITA2Go** how to sort and present <index-see> and <index-see-also> terms.

*In this section:*

§16.5.6.1 [Identifying See and See also index references](#) on page 254

§16.5.6.2 [Specifying level breaks for See and See also index entries](#) on page 254

§16.5.6.3 [Choosing where to sort See also index references](#) on page 255

### 16.5.6.1 Identifying See and See also index references

To configure *See* and *See also* entries in the index, you can specify the words used as starting terms in those entries. **DITA2Go** uses this information for sorting (see §16.5.6.3 [Choosing where to sort See also index references](#) on page 255) and for merging (see §16.11 [Setting up a dynamic modular Help system](#) on page 280).

To specify the words to use for *See* and *See also* in the 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
```

**DITA2Go** 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, **DITA2Go** redirects both *See* and *See also* links to their targets in the index itself; see §19.7.6 [Redirecting See and See also index entries](#) on page 370. 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.

### 16.5.6.2 Specifying level breaks for See and See also index entries

By default, **DITA2Go** forces an index level break for a *See* or *See also* index entry. For example:

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 §16.5.4 [Treating commas as potential index level separators](#) on page 252.

### 16.5.6.3 Choosing where to sort *See also* index references

By default, **DITA2Go** 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.

## 16.5.7 Specifying index link destinations for HTML-based Help

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 16-1](#), depending on the setting you specified for `KeywordRefs`.

**Table 16-1 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 PI marker.
Para	<code>topicfile.htm#objectID</code>	Start of the paragraph containing the PI marker; use when other index markers occur in the same paragraph.
File	<code>topicfile.htm</code>	Start of the file containing the PI marker; use if all index entries 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`, **DITA2Go** generates index link destinations of the following form:

```
topicfile.htm#objectID
```

where *objectID* is an internally generated ID number.

*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
```

## 16.5.8 Customizing index sort order

For stand-alone (unmerged) HTML Help and for OmniHelp, as well as for non-Help HTML (see §14.8.6 [Configuring index features for HTML output](#) on page 211), **DITA2Go** can control the order of index entries and subentries. Merged HTML Help requires a binary index, which ignores **DITA2Go** settings. JavaHelp and Oracle Help tend to ignore any sort order **DITA2Go** produces.

*In this section:*

§16.5.8.1 [Listing characters to ignore in index sort order](#) on page 256

§16.5.8.2 [Choosing case sensitivity of indexed terms](#) on page 257

§16.5.8.3 [Specifying index sort type and locale](#) on page 257

*See also:*

§16.5.6.3 [Choosing where to sort See also index references](#) on page 255

### 16.5.8.1 Listing characters to ignore in index sort order

To specify which characters **DITA2Go** 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
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 **DITA2Go** ignores the following characters:

- anywhere in an entry:
  - hyphen
  - [ ] 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`, **DITA2Go** 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=
```

### 16.5.8.2 Choosing case sensitivity of indexed terms

If your DITA 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.

### 16.5.8.3 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, **DITA2Go** does support multibyte sorting when you specify the index locale.

If you use **DITA2Go** to produce HTML Help in an Asian or Cyrillic language, also specify the Help-file language; see §18.12 [Generating HTML Help in non-Western languages](#) on page 344.



## 16.6 Providing related-topic links for Help systems

DITA2Go 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:*

- § 16.6.1 [Understanding related-topic links](#) on page 258
- § 16.6.2 [Understanding how ALinks work](#) on page 259
- § 16.6.3 [Understanding how KLinks work](#) on page 259
- § 16.6.4 [Adding related-topic link keywords in DITA XML](#) on page 260
- § 16.6.5 [Adding ALink and KLink jumps in DITA XML](#) on page 261
- § 16.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 262
- § 16.6.7 [Specifying ALink and KLink list-link destinations](#) on page 262

### 16.6.1 Understanding related-topic links

Related-topic links you can produce with **DITA2Go** 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. ALink keywords are not ordinarily visible to users, except in OmniHelp (see § 19.8 [Providing related-topic links in OmniHelp](#) on page 370).

*Link keywords* 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.

*Related-topic jumps* You can generate ALinks from DITA related-links and reltables; or, you can insert ALinks in DITA XML with **ALink** PI markers. Use one or the other method; do not use both.

*Bullet-proof links* Why use ALinks and KLinks if your document already includes cross references, *See also* 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 *require* the presence of any topic. Therefore, you can do the following without disturbing related-topic links:

*Add a topic:* Existing ALink and KLink jumps automatically pick up any relevant link keywords in the new topic.

*Remove a topic:* 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.

*Run-time activation* ALinks and KLinks are especially useful if you expect to merge Help projects (see § 16.11 [Setting up a dynamic modular Help system](#) on page 280), for the following reasons:

- 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.

*KLinks access  
merged topics*

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.

## 16.6.2 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 **DITA2Go**-generated Help systems support ALinks:

WinHelp 4  
HTML Help  
OmniHelp  
Oracle Help for Java

*ALink keyword*

You insert an ALink keyword (via **ALink** PI marker) at the start of a topic, to accomplish the following:

- 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.

The ALink set is identified by the ALink keyword. In effect, you label the topic with an ALink keyword.

*ALink jump*

At the start of some other topic, you insert an ALink jump (via **ALinkJump** PI marker) 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 §16.6.4.1 [Adding related-topic link keywords via PI markers](#) on page 260.

For a similar approach in HTML Help and Oracle Help for Java, see §16.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 262.

## 16.6.3 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 **DITA2Go**-generated Help systems nominally support KLinks, though only for index entries that meet assorted restrictions:

HTML Help  
OmniHelp  
WinHelp 4

You insert at the start of a topic a KLink jump (via **KLinkJump** PI marker) 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 DITA2Go-generated Help systems where they are nominally supported.

## 16.6.4 Adding related-topic link keywords in DITA XML

*In this section:*

§16.6.4.1 [Adding related-topic link keywords via PI markers](#) on page 260

§16.6.4.2 [Adding related-topic keywords via format properties](#) on page 260

*See also:*

§16.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 262

### 16.6.4.1 Adding related-topic link keywords via PI markers

You can use DITA **ALink** PI markers to insert ALink keywords. The content of the PI 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 **ALink** PI marker, separated by semicolons.

You can provide **ALink** PI markers to use for ALink keywords by inserting PIs in DITA XML with the following content:

```
ALink="keyword"
```

### 16.6.4.2 Adding related-topic keywords via format properties

You can assign a property to a format to make the text of every instance of that format act as a related-topic keyword. Create a special `@outputclass` 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 an inline element with a special `@outputclass` to the word, and in the configuration file assign property `AKey` to the resulting format:

```
[HelpStyles]
ALinkCharFmt = AKey
```

Or, you can insert a block element containing only the keyword, add a special `@outputclass`, and also assign property `Delete` to the resulting format:

```
[HelpStyles]
ALinkParaFmt = AKey Delete
```

See §17.10 [Creating related-topic links in WinHelp](#) on page 303.

*HTML-based  
Help*

For HTML-based Help you must use a block element (as opposed to an inline element) a for related-topic keywords. For example, you can put an ALink keyword in a block element by itself, add a special `@outputclass`, and in the configuration file assign

property `ALink` to the resulting 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
```

## 16.6.5 Adding ALink and KLink jumps in DITA XML

You can use DITA PI markers to insert ALink and KLink jumps in your document, or you can direct **DITA2Go** to generate ALinks from related links; see §13.5 [Generating associative links for Help output](#) on page 192.

*In this section:*

§16.6.5.1 [Configuring ALink jumps](#) on page 261

§16.6.5.2 [Configuring KLink jumps](#) on page 261

*See also:*

§16.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 262

### 16.6.5.1 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 DITA XML, insert a PI marker with content like the following:

```
ALinkJump="keyword"
```

If you specify multiple ALink keywords (which you can for OmniHelp or HTML Help), separate the identifiers with semicolons (no spaces!). For example:

```
ALinkJump="curry;chicken;turmeric"
```

### 16.6.5.2 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.

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 DITA XML, insert a hypertext **Go to URL** marker with content like the following:

```
KLinkJump="index term1;index term2;..."
```

Separate index terms from each other with semicolons (no spaces). Index terms can contain spaces

*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.

### 16.6.6 Creating target-and-jump ALinks for HTML-based Help

You can insert ALink information in DITA XML 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 PI markers has this effect. Whenever the OmniHelp navigation control is set to <b>Related</b> , if the currently displayed topic contains an ALink keyword PI marker, the navigation pane displays a list of links to all other topics that contain <b>ALink</b> PI markers with the same keyword.
<i>HTML Help, Oracle Help for Java</i>	For HTML Help or Oracle Help for Java, you use a block element with a special @outputclass to specify ALink keywords, and supply macro code to surround the resulting paragraph for the ALink jump. When you assemble ALink jumps using macros, you are not making use of any <b>DITA2Go</b> code to interpret the alink 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 §18.7.4 <a href="#">Rolling your own macros for ALink jumps in HTML Help</a> on page 328.
<i>Oracle Help for Java</i>	For Oracle Help for Java, the ALink jump code creates a hotspot; see §20.10 <a href="#">Creating ALinks for Oracle Help</a> on page 409.

### 16.6.7 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 §20.10 [Creating ALinks for Oracle Help](#) on page 409.

## 16.7 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:*

§16.7.1 [Assigning secondary windows for WinHelp](#) on page 262

§16.7.2 [Assigning secondary windows for HTML-based Help](#) on page 263

### 16.7.1 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 §17.8.6 [Specifying jumps to secondary windows in WinHelp](#) on page 302.

## 16.7.2 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 §16.8 [Creating pop-up topics for Help systems](#) on page 263. For OmniHelp, you can include optional window parameters in the assignment.

*See also:*

§18.8 [Using secondary windows in HTML Help](#) on page 332

§19.9 [Jumping to secondary windows in OmniHelp](#) on page 370

§20.8.3 [Jumping to secondary windows in JavaHelp or Oracle Help](#) on page 408

§28.4 [Creating jumps to particular windows for HTML](#) on page 550

## 16.8 Creating pop-up topics for Help systems

You can use **DITA2Go** to create pop-up topics in any of the Help formats.

*In this section:*

§16.8.1 [Understanding pop-up hotspots, links, and topics](#) on page 263

§16.8.2 [Defining a pop-up hotspot](#) on page 264

§16.8.3 [Displaying a topic in a pop-up window](#) on page 264

### 16.8.1 Understanding pop-up hotspots, links, and topics

In DITA XML, you delimit a pop-up *hotspot* by applying a dedicated `@outputclass` 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 §20.8.1.4 [Specifying window-access object properties](#) on page 405.

## 16.8.2 Defining a pop-up hotspot

To define a hotspot, in DITA XML apply a dedicated `@outputclass` 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.

## 16.8.3 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 §17.8 [Creating jumps and pop-ups for WinHelp](#) on page 299.

*HTML-based Help* If your HTML-based Help system has a **DITA2Go**-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 §27.4 [Extracting files](#) on page 528.

*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 DITA hypertext **alert** PI marker embedded in the hotspot.

See §18.5 [Creating pop-ups for HTML Help](#) on page 322.

*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:

§19.9 [Jumping to secondary windows in OmniHelp](#) on page 370

§20.8 [Defining windows for JavaHelp or Oracle Help](#) on page 403.

## 16.9 Including expandable sections in Help topics

For OmniHelp and HTML Help (and for HTML and XHTML), you can use a combination of JavaScript and **DITA2Go** macros to create one or more expandable drop-down sections in a topic.

*In this section:*

§16.9.1 [Understanding DITA2Go expandable drop-down sections](#) on page 265

§16.9.2 [Setting up expandable sections for your document](#) on page 265

§16.9.3 [Delimiting expandable drop-down sections](#) on page 266

§16.9.4 [Configuring drop-down links](#) on page 268

§16.9.5 [Configuring drop-down blocks](#) on page 271

§16.9.6 [Providing CSS for drop-down links and blocks](#) on page 271

§16.9.7 [Deploying JavaScript code for drop-down sections](#) on page 271

§16.9.8 [Emulating Web Works Publisher drop-down hotspots](#) on page 275

## 16.9.1 Understanding DITA2Go 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 **DITA2Go** output, the link can be based on any of the following:

- a special block-element `@outputclass` dedicated to drop-down links
- an existing block element 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.

**DITA2Go** provides built-in macros to use for drop-down sections, and settings to enable and deploy the macros. To use the built-in **DITA2Go** 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 **DITA2Go** 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.

## 16.9.2 Setting up expandable sections for your document

To enable expandable drop-down sections in the HTML output from your DITA 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 block-element `@outputclass` for material you want to allow users to expand.
- Assign property `DropDown` to the resulting format.
- Let **DITA2Go** supply the drop-down links.

For example, suppose you apply `@outputclass="ExpandThis"` to each paragraph to be expanded. You would include the following setting in your project configuration file:

```
[HTMLParaStyles]
ExpandThis = DropDown
```

With this setting, **DITA2Go** 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.



<i>Multiple drop-down paragraphs</i>	To include multiple paragraphs in an expandable section, or to include material in some other format, surround the material to be expanded with <b>Code</b> PI markers in otherwise empty paragraphs; see §16.9.3 <a href="#">Delimiting expandable drop-down sections</a> on page 266.
<i>Customized drop-down section</i>	<p>To position drop-down links yourself, or to use existing text in your document for the links, or both:</p> <ul style="list-style-type: none"> <li>• Place a paragraph with a dedicated <code>@outputclass</code> 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.</li> <li>• 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 §16.9.3.1 <a href="#">Delimiting drop-down links and blocks with paragraph formats</a> on page 266.</li> <li>• For any existing link text that is <i>not</i> in the drop-down link format, surround the link text with <b>Code</b> PI 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 §16.9.3.2 <a href="#">Delimiting drop-down links and blocks with markers</a> on page 267.</li> <li>• Specify display options for drop-down links and blocks; see §16.9.4 <a href="#">Configuring drop-down links</a> on page 268 and §16.9.5 <a href="#">Configuring drop-down blocks</a> on page 271.</li> </ul>

## 16.9.3 Delimiting expandable drop-down sections

Creating drop-down links and expandable blocks involves surrounding material in your document with built-in **DITA2Go** 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:*

§16.9.3.1 [Delimiting drop-down links and blocks with paragraph formats](#) on page 266

§16.9.3.2 [Delimiting drop-down links and blocks with markers](#) on page 267

### 16.9.3.1 Delimiting drop-down links and blocks with paragraph formats

The following settings use built-in **DITA2Go** 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; **DITA2Go** generates the link paragraph.

To expand a single-paragraph block using a **DITA2Go**-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 16-2 shows the effects of these format properties.

**Table 16-2 Effects of drop-down format properties**

Drop-down style	Format property	Effect
DITA2Go-supplied link, single-paragraph block	DropDown	Treats the current paragraph as the block to be expanded. Places macros <\$DropLinkPara> and <\$DropBlockStart> before the paragraph, <\$DropBlockEnd> after the paragraph.
Variable-text link, single-paragraph block	DropDownLink	Treats the current paragraph or character span as the link. Places macro <\$DropLinkStart> before the text in the paragraph or character span, <\$DropLinkEnd> after the text.
	DropDownBlock	Treats the current paragraph as the block to be expanded. Places macro <\$DropBlockStart> before the paragraph, outside its tags, and <\$DropBlockEnd> after the closing tags of the paragraph.
Variable-text link, multiple-paragraph block	DropDownStart	Treats the current paragraph as the link, and the next paragraph or paragraphs as the block. Places macro <\$DropLinkStart> before the text in the current paragraph, <\$DropLinkEnd> after the text, then <\$DropBlockStart> after the paragraph.
	DropDownEnd	Treats the current paragraph as the last paragraph in the block to be expanded. Places macro <\$DropBlockEnd> after the current paragraph.

### 16.9.3.2 Delimiting drop-down links and blocks with markers

You can use **Code** PI markers to surround drop-down links and expandable blocks with built-in **DITA2Go** macro code.

To delimit links and blocks with **Code** markers, place a **Code** PI 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 §16.9.4.1 [Specifying the type of link for drop-down sections](#) on page 268.)

- 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 drop-down content tags. Put the **Code** PI 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 PI markers, and assign the following property to the format:

```
[HTMLParaStyles]
CodeMarkerFmt = Raw
```

See §30.2.4 [Stripping paragraph properties](#) on page 568.

## 16.9.4 Configuring drop-down links

By default, **DITA2Go** 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:*

- §16.9.4.1 [Specifying the type of link for drop-down sections](#) on page 268
- §16.9.4.2 [Configuring icons for drop-down links](#) on page 269
- §16.9.4.3 [Configuring buttons for drop-down links](#) on page 269
- §16.9.4.4 [Configuring text for drop-down links](#) on page 270
- §16.9.4.5 [Modifying code for drop-down links](#) on page 270

### 16.9.4.1 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`, **DITA2Go** 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 §16.9.4.2 [Configuring icons for drop-down links](#) on page 269.

When `DropLinkType=Button`, **DITA2Go** inserts a button in each drop-down link paragraph. You can specify the label on the button; see §16.9.4.3 [Configuring buttons for drop-down links](#) on page 269.

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 §16.9.4.4 [Configuring text for drop-down links](#) on page 270.

Regardless of link type, **DITA2Go** 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, **DITA2Go** creates a paragraph for the link, just before each drop-down block.

By default, **DITA2Go** includes JavaScript code that works for all types of drop-down links. However, you can choose to have **DITA2Go** 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 must use the same type of drop-down link, because the type determines the JavaScript that is included or referenced in the output. See §16.9.7 [Deploying JavaScript code for drop-down sections](#) on page 271.

#### 16.9.4.2 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.

**DITA2Go** 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 **DITA2Go** write these default icon files to the project directory:

```
[DropDowns]
; WriteDropIconFiles = No (default) or Yes (write to project
; directory)
WriteDropIconFiles = Yes
```

When `WriteDropIconFiles=Yes`, **DITA2Go** 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 §16.9.4.5 [Modifying code for drop-down links](#) on page 270.

#### 16.9.4.3 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 §16.9.4.5 [Modifying code for drop-down links](#) on page 270.

#### 16.9.4.4 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 **DITA2Go** 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 §16.9.3 [Delimiting expandable drop-down sections](#) on page 266.

To change the macro code that displays fixed text, see §16.9.4.5 [Modifying code for drop-down links](#) on page 270.

#### 16.9.4.5 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 **DITA2Go** 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: §16.9.6 [Providing CSS for drop-down links and blocks](#) on page 271:

```
[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>
```

## 16.9.5 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.*

## 16.9.6 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 §31 [Setting up CSS for HTML](#) on page 591.

## 16.9.7 Deploying JavaScript code for drop-down sections

Based on the configuration settings you specify for drop-down links and blocks, **DITA2Go** 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:*

- §16.9.7.1 [Naming the JavaScript macro for drop-down sections](#) on page 272
- §16.9.7.2 [Locating JavaScript code for drop-down sections](#) on page 272
- §16.9.7.3 [Directing DITA2Go to write drop-down JavaScript code](#) on page 273
- §16.9.7.4 [Inspecting the JavaScript code for drop-down sections](#) on page 273

### 16.9.7.1 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, **DITA2Go** includes either the composite JavaScript code (see §16.9.4.1 [Specifying the type of link for drop-down sections](#) on page 268) or one of four built-in versions of this macro; see §16.9.7.4 [Inspecting the JavaScript code for drop-down sections](#) on page 273.

### 16.9.7.2 Locating JavaScript code for drop-down sections

By default, for most output types **DITA2Go** 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, **DITA2Go** includes the JavaScript code in viewer files `ohctrl.js` and `ohmain.js`. For any output type, you can direct **DITA2Go** 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`, **DITA2Go** 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 **DITA2Go** writes the output HTML file.

When `DropJSLocation=None`, **DITA2Go** 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 §37.9.2 [Invoking macros at predetermined points in output](#) on page 710.

When `DropJSLocation=filename`, **DITA2Go** places the following reference in the `<head>` section:

```
<script language="JavaScript" type="text/javascript"
src="<$DropJSLocation>"></script>
```

Macro `$DropJSLocation` is expanded when **DITA2Go** 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.

### 16.9.7.3 Directing DITA2Go to write drop-down JavaScript code

By default, **DITA2Go** does not create an external version of the drop-down JavaScript code. To have **DITA2Go** 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`, **DITA2Go** assumes that the file you specified for `DropJSLocation` is an existing JavaScript library that you do not want overwritten.

When `WriteDropJSFile=Yes`, **DITA2Go** 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 §16.9.7.2 [Locating JavaScript code for drop-down sections](#) on page 272). And while any path information included in *filename* is used in the link in the `<head>` section, **DITA2Go** 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.

### 16.9.7.4 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:*

§16.9.7.4.1 [JavaScript code when UseCompositeDropJS=Yes](#) on page 273

§16.9.7.4.2 [JavaScript code when DropLinkType=Icon](#) on page 274

§16.9.7.4.3 [JavaScript code when DropLinkType=Button](#) on page 274

§16.9.7.4.4 [JavaScript code when DropLinkType=Text](#) on page 275

*See also:*

§16.9.4.1 [Specifying the type of link for drop-down sections](#) on page 268.

#### 16.9.7.4.1 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"
    }
}

```

#### 16.9.7.4.2 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"
    }
}

```

#### 16.9.7.4.3 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>"
}
}
}

```

#### 16.9.7.4.4 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"
}
}
}

```

## 16.9.8 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 **DITA2Go** macros, either with or without built-in **DITA2Go** drop-down controls. Jim Owens has kindly allowed us to present the macros he developed for this purpose.

*In this section:*

§16.9.8.1 [Creating drop-down hotspots with DITA2Go controls and macros](#) on page 275

§16.9.8.2 [Creating drop-down hotspots with DITA2Go macros only](#) on page 276

### 16.9.8.1 Creating drop-down hotspots with DITA2Go controls and macros

The following settings use **DITA2Go** 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

```

### 16.9.8.2 Creating drop-down hotspots with DITA2Go 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

```

## 16.10 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:*

§16.10.1 [Understanding how CSH works](#) on page 278

§16.10.2 [Specifying CSH mappings](#) on page 278

*See also:*

§18.11 [Setting up CSH for HTML Help](#) on page 340

§19.11 [Setting up CSH for OmniHelp](#) on page 375

§20.12 [Setting up CSH for JavaHelp or Oracle Help](#) on page 410

## 16.10.1 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; if 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, **DITA2Go** 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; **DITA2Go** 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.

## 16.10.2 Specifying CSH mappings

To provide CSH when **DITA2Go** generates Help files for your project:

1. **Give each target topic a `<data />` element that contains a symbolic ID.**  
In your DITA document, insert a `<data />` element in each topic that will be the target of a call from the application program. The name is `topicalias` and the value

is the symbolic ID for the topic. Insert a separate `<data />` element with a unique symbolic ID for each call from the application. Put the `<data />` element in the topic, after the root and before the title. For example:

```
<topic id="framistans">
  <data name="topicalias" value="IDH_framistan" />
  <title>Framistan maintenance</title>
  ...
</topic>
```

**DITA2Go** also supports Omni Systems **TopicAlias** PI markers of the form:

```
<?dthtm TopicAlias="IDH_about" ?>
```

and native FrameMaker DITA CSH PI markers of the form:

```
<?FM MARKER [TopicAlias] IDH_about?>
```

as well as DITA-FMx `<data />` elements of the form:

```
<data datatype="fm:marker" name="TopicAlias" value="IDH_about" />
```

2. **Create or obtain a map file** (possibly except JavaHelp and Oracle Help for Java; see §20.12 [Setting up CSH for JavaHelp or Oracle Help](#) on page 410).
3. **Specify prefixes that identify CSH links** (HTML Help or OmniHelp).  
List topic-name prefixes in the configuration file, to identify **TopicAlias** PI markers intended for CSH use. If you do not specify any prefixes, all **TopicAlias** PI 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** PI 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
```

**DITA2Go** creates a CSH link destination from each **TopicAlias** PI marker whose name starts with one of the prefixes you specified in [Step 3](#), or all the **TopicAlias** PI markers if you did not specify any prefixes. Make sure the symbolic IDs in the **TopicAlias** PI markers are spelled the same way as in the map file.

By default, **DITA2Go** removes punctuation and spaces from the **TopicAlias** PI 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
```

## 16.11 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 **DITA2Go** 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	§17.2.8 <a href="#">Providing multiple .hlp files</a> on page 285
MS HTML Help	§18.14 <a href="#">Mapping and merging CHM files</a> on page 348
OmniHelp	§19.12 <a href="#">Merging OmniHelp projects</a> on page 377
JavaHelp, Oracle Help for Java	§20.11 <a href="#">Merging JavaHelp or Oracle Help systems</a> on page 410
Eclipse Help	§21.6 <a href="#">Merging Eclipse Help projects</a> on page 423

# 17 Generating WinHelp

---

**DITA2Go** 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:

- §17.1 [Obtaining tools for WinHelp](#) on page 281
- §17.2 [Setting up a WinHelp project](#) on page 281
- §17.3 [Converting text](#) on page 286
- §17.4 [Converting cross references](#) on page 288
- §17.5 [Converting tables to WinHelp RTF](#) on page 290
- §17.6 [Managing graphics for WinHelp](#) on page 292
- §17.7 [Configuring WinHelp topics](#) on page 294
- §17.8 [Creating jumps and pop-ups for WinHelp](#) on page 299
- §17.9 [Invoking WinHelp macros](#) on page 302
- §17.10 [Creating related-topic links in WinHelp](#) on page 303
- §17.11 [Configuring index entries for WinHelp](#) on page 305
- §17.12 [Configuring contents for WinHelp](#) on page 306
- §17.13 [Creating browse sequences](#) on page 310

*See also:*

- §15 [Converting to print RTF](#) on page 219, for information about settings that work the same way for print RTF and for WinHelp.
- §16 [Producing on-line Help](#) on page 243, for information about configuring contents and index, providing related-topics links, supporting context-sensitive help, and merging help projects.

## 17.1 Obtaining tools for WinHelp

To generate WinHelp you need Microsoft Help Workshop, `hcw.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 §16.1 [Weighing Help-system alternatives](#) on page 243.

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.

## 17.2 Setting up a WinHelp project

When you specify WinHelp for output, **DITA2Go** produces the `.hbj`, the `.cnt`, the `.wmf` graphics, and the WinHelp-coded `.rtf` files. You open the `.hbj` in Help Workshop, and click **Compile**. You can view the result in Help Workshop.

*In this section:*

- §17.2.1 [Setting up a WinHelp project](#) on page 282
- §17.2.2 [Deciding where to locate configuration settings](#) on page 282
- §17.2.4 [Deciding whether to regenerate the WinHelp project file](#) on page 282
- §17.2.5 [Accommodating platform differences](#) on page 283
- §17.2.6 [Setting basic WinHelp options in the configuration file](#) on page 284
- §17.2.7 [Handling page breaks and section breaks](#) on page 285
- §17.2.8 [Providing multiple .hlp files](#) on page 285
- §17.2.9 [Integrating WinHelp from RoboHelp](#) on page 285
- §17.2.10 [Compiling a WinHelp project](#) on page 285
- §17.2.11 [Checking WinHelp RTF files for DITA2Go version](#) on page 286

## 17.2.1 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 DITA document is located.
2. Use a text editor to specify settings in configuration file `_d2winhelp.ini` (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

## 17.2.2 Deciding where to locate configuration settings

When you set up a WinHelp project, if configuration file `_d2winhelp.ini` is not already present in the project directory, you must copy this file from your **DITA2Go** `config\local` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29).

*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	<code>_d2winhelp.ini</code>	Current project directory
All WinHelp projects	<code>local_d2winhelp_config.ini</code>	<code>%omsyshome%\d2g\local\config\</code>

See §39.4 [Deciding which configuration file to edit](#) on page 734.

## 17.2.3 Preparing a document for conversion to WinHelp

To generate WinHelp from a DITA 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.
- Provide versions of graphics in BMP or WMF format.

## 17.2.4 Deciding whether to regenerate the WinHelp project file

When you use **DITA2Go** to generate WinHelp, **DITA2Go** writes an `.hproj` project file during set-up, and rewrites it later only under certain conditions.

To specify whether **DITA2Go** should generate the `.hproj` 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 .hpf file is present, <b>DITA2Go</b> overwrites it.
No	<b>DITA2Go</b> does not overwrite the .hpf file.
(none)	If the configuration file contains no WriteHelpProjectFile setting at all, <b>DITA2Go</b> writes an .hpf file, but <i>only if the .hpf file is not already present</i> .

**DITA2Go** closes the .hpf file after writing it; however, if you had the .hpf file open in Help Workshop when **DITA2Go** 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 .hpf 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 **DITA2Go** from overwriting them by setting WriteHelpProjectFile=No.

## 17.2.5 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 **DITA2Go** 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, **DITA2Go** has a setting for Bristol HyperHelp: HyperHelp=Yes. Unfortunately, HyperHelp has trouble with WMF graphics. You must use a graphics conversion program to convert all WMF graphics to BMP graphics. 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
```

## 17.2.6 Setting basic WinHelp options in the configuration file

When you set up a WinHelp project, you must provide your own WinHelp.hpj file, 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 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 §17.12.2.1 [Understanding WinHelp contents level numbers](#) on page 307.

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
```

To have **DITA2Go** 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 §44.6 [Assembling files for distribution](#) on page 792.

## 17.2.7 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. **DITA2Go** creates this sequence automatically when you specify `[HelpStyles]ParaFmt=Topic` (see §17.7.2 [Assigning properties to formats for topics and hotspots](#) on page 295).

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. 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
```

## 17.2.8 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:

§17.4.2 [Specifying cross-reference destination files](#) on page 289

§17.12.2.5 [Referencing multiple help files from contents](#) on page 309

As an alternative, you can provide a main .hlp file with contents entries that link to other .hlp files; see §16.11 [Setting up a dynamic modular Help system](#) on page 280.

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*.

## 17.2.9 Integrating WinHelp from RoboHelp

If you have RoboHelp installed on your system, you can integrate a WinHelp DLL created with RoboHelp into your **DITA2Go**-produced WinHelp project.

1. Use RoboHelp to generate a WinHelp 2000 project of any size, even one short file.
2. Open the resulting .hpj 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 .hpj file produced by **DITA2Go**.

## 17.2.10 Compiling a WinHelp project

To compile the RTF files **DITA2Go** 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 **DITA2Go** 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 specify the following options in the configuration file, **DITA2Go** automatically runs the WinHelp compiler after generating output files:

```
[Automation]
; CompileHelp = No (default, run Help compiler separately), or Yes
CompileHelp=Yes
```

To have **DITA2Go** 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 §44.6 [Assembling files for distribution](#) on page 792.

*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 (.hbj) file. For example:

```
[OPTIONS]
BMROOT=..\MyGraphics
BMROOT=..\Test\Graphics
```

## 17.2.11 Checking WinHelp RTF files for DITA2Go version

If you recently installed a **DITA2Go** upgrade or beta version, after you run **DITA2Go**, 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 **DITA2Go** drmfif.dll and dwrtf.dll components that were used to create the RTF file. See §A.1.5 [Check your version of DITA2Go](#) on page 820.

## 17.3 Converting text

*In this section:*

§17.3.1 [Suppressing unwanted paragraphs](#) on page 287

§17.3.2 [Converting autonumbers](#) on page 287

§17.3.3 [Replacing paragraph or character content](#) on page 287

§17.3.4 [Converting footnotes](#) on page 288

### 17.3.1 Suppressing unwanted paragraphs

To prevent text from appearing in RTF output:

1. Assign a special paragraph format to 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.

### 17.3.2 Converting autonumbers

By default, **DITA2Go** includes paragraph autonumbers as text in WinHelp output. To omit autonumbers:

```
[HelpOptions]
; WriteAnums = Yes (default) or No (omit autonumbers)
WriteAnums = No
```

When `WriteAnums=Yes` (the default), **DITA2Go** adds numbering prefixes to text paragraphs, according to numbering properties you assign to paragraph formats (see §8.5 [Configuring output numbering properties](#) on page 146) and format definitions you specify in a format configuration file (see §7.5 [Understanding text output formats](#) on page 119).

When `WriteAnums=No`, **DITA2Go**-supplied numbering is omitted.

### 17.3.3 Replacing paragraph or character content

You can direct **DITA2Go** 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 **DITA2Go User's Guide**, to replace page numbers in keyword indexes with bitmap graphics.

## 17.3.4 Converting footnotes

**DITA2Go** can convert a DITA 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.

## 17.4 Converting cross references

By default, **DITA2Go** converts cross references into jumps, including interfile jumps where needed.

*In this section:*

- §17.4.1 [Creating help context markers](#) on page 288
- §17.4.2 [Specifying cross-reference destination files](#) on page 289
- §17.4.3 [Specifying cross-reference jump destinations](#) on page 289
- §17.4.4 [Specifying WinHelp options for cross-reference formats](#) on page 290
- §17.4.5 [Limiting cross-reference text](#) on page 290

### 17.4.1 Creating help context markers

**DITA2Go** 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 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) **DITA2Go** uses internally generated object IDs instead of the full cross-reference text.

## 17.4.2 Specifying cross-reference destination files

If all your cross references are to sources within the same `.hlp` file, **DITA2Go** 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* **DITA2Go** checks `[HelpXrefFiles]` for each cross reference to another file; if the file name is not listed, **DITA2Go** 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 §17.12.2.5 [Referencing multiple help files from contents](#) on page 309.

## 17.4.3 Specifying cross-reference jump destinations

**DITA2Go** 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 = dita2go
```

#### 17.4.4 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 **DITA2Go** 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, **DITA2Go** would render any cross reference that uses the *Heading & Page* format as plain text rather than as a link. **DITA2Go** would also delete any cross reference that uses the *Page* format.

#### 17.4.5 Limiting cross-reference text

DITA 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 **DITA2Go** 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 §17.8.2 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 299.

### 17.5 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:*

§17.5.1 [Positioning tables and table titles](#) on page 290

§17.5.2 [Adjusting table appearance](#) on page 291

§17.5.3 [Converting table rows to topics and table cells to pop-ups](#) on page 292

#### 17.5.1 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 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.

## 17.5.2 Adjusting table appearance

Before you attempt the fine tuning described in this section:

- To specify output formats and general options for DITA table types, see §6.9 [Specifying formats and options for tables](#) on page 103.
- To map table elements to output formats using attributes, see §6.3.2 [Mapping table outputclass attributes to formats](#) on page 90.
- To define properties for each table format, see §7.7 [Configuring table output formats](#) on page 129.

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
```

`TableRules` and `TableFill` determine whether corresponding values set in a table-format configuration file take effect:

- If `TableRules=None`, **DITA2Go** does not write borders.
- If `TableRules=Box, Double, Thick, or Shadow`, **DITA2Go** ignores the corresponding table-format configuration values.
- If `TableFill=None` or `ColorOnly`, **DITA2Go** ignores the table-format configuration settings for shading.

When `ForceTableLineBreaks=Yes`, **DITA2Go** 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** **DITA2Go** has two settings available to fine tune the cell size. 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, **DITA2Go** 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 §15.6 [Converting tables to print RTF](#) on page 232.

### 17.5.3 Converting table rows to topics and table cells to pop-ups

**DITA2Go** 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, **DITA2Go** 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 §17.7.1 [Creating WinHelp topics](#) on page 294), 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
```

## 17.6 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.

To exclude graphics entirely, or display only graphics file names, see §40.2.2.3 [Excluding graphics from RTF output](#) on page 751.

*In this section:*

§17.6.1 [Choosing a graphics format for WinHelp](#) on page 293

§17.6.2 [Displaying graphics in pop-ups for WinHelp](#) on page 293

*See also:*

§17.8.2.3 [Embedding hotspots in graphics for WinHelp](#) on page 300

## 17.6.1 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.

For non-Windows WinHelp, created via Altura on Macintosh or Bristol HyperHelp on UNIX, only BMP is supported; see §17.2.5 [Accommodating platform differences](#) on page 283.

## 17.6.2 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. **DITA2Go** provides two ways to accomplish this:

- Insert hypertext PI 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 PI markers for pop-up graphics (see §17.8.4 [Using hypertext links for jumps and pop-ups](#) on page 301):

1. Embed a **HyperAnchor** PI marker with a unique name in the paragraph that holds the anchor for the graphic.
2. Embed a corresponding **HyperJump** PI 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:

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
```

## 17.7 Configuring WinHelp topics

To produce the organization and navigation features of WinHelp, you assign topic and hotspot properties to paragraph, character, and cross-reference formats assigned to DITA elements. The first property assigned to each format identifies the role that all text in that format will play in the Help system.

*In this section:*

- §17.7.1 [Creating WinHelp topics](#) on page 294
- §17.7.2 [Assigning properties to formats for topics and hotspots](#) on page 295
- §17.7.3 [Configuring topic titles for WinHelp](#) on page 297

### 17.7.1 Creating WinHelp topics

To create topics, in the configuration file assign property `Topic` to the 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 §17.5.3 [Converting table rows to topics and table cells to pop-ups](#) on page 292.

The format to which you assign property `Topic` becomes the title of the topic in WinHelp; see §17.7.3 [Configuring topic titles for WinHelp](#) on page 297.

You can create three kinds of WinHelp topics with **DITA2Go**:

- 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 DITA XML, 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 §17.7.2 [Assigning properties to formats for topics and hotspots](#) on page 295 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 §17.8.1 [Configuring pop-up topics](#) on page 299.

## 17.7.2 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 §17.7.1 [Creating WinHelp topics](#) on page 294
- Hotspot Delimiting character (or paragraph) format; see §17.8.2 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 299.

[Table 17-1](#) shows which properties you can assign to formats as starting and following properties for topics and hotspots. [Table 17-2](#) shows the effects of the properties you assign.

**Table 17-1 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 <sup>1</sup> Resume SpKey Suffix
	Topic	AKey Browse Build Contents Delete Key Macro MakeRef NoScroll NoTitle NoXScroll Refer Scroll Slide SpKey Suffix TitleSuf Window XScroll

**Table 17-1 Starting and following format properties for topics and hotspots**

Type	Starting	Following
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 <sup>2</sup> Key ParaLink PrevRef Resume SpKey Suffix Uline <sup>2</sup>
	Replace	AKey Key Resume SpKey Suffix
	Resume	Delete
	SpKey	AKey Delete Key Resume
<sup>1</sup> Within current topic only.		<sup>2</sup> Applied to cross references or hypertext links.,

**Table 17-2 Effects of format properties on topics and hotspots**

Property	Effect
AKey	Adds the topic title as an “A” footnote. See §17.10.2 <a href="#">Adding ALinks and KLinks with PI markers</a> on page 304.
Browse	Includes the topic in the browse sequence; <b>DITA2Go</b> creates the “+” footnote. See §17.13 <a href="#">Creating browse sequences</a> on page 310.
Build	Specifies a “build tag” for the topic, defined in the .hlp.j 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> ; <b>DITA2Go</b> creates the “*” footnote.
Contents	Includes the topic (as a page, book, or both) in the WinHelp .cnt file produced by <b>DITA2Go</b> . See §17.12 <a href="#">Configuring contents for WinHelp</a> on page 306.
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 §17.3.1 <a href="#">Suppressing unwanted paragraphs</a> on page 287.
File	Directs the jump or pop-up to a topic in a different .hlp file.
Green	Makes a hotspot green and underlined. See §17.8.2.1 <a href="#">Configuring jump vs. pop-up hotspots</a> on page 300.
Key	Adds the topic title as a keyword, so it appears in the WinHelp index. See §17.10.3 <a href="#">Adding related-topic keywords with formats</a> on page 304.
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.
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> ; <b>DITA2Go</b> creates the “!” footnote. See §17.9 <a href="#">Invoking WinHelp macros</a> on page 302.
MakeRef	Creates a unique reference tag for the current paragraph, to use as a pop-up or jump destination for a PrevRef paragraph; see §17.6.2 <a href="#">Displaying graphics in pop-ups for WinHelp</a> on page 293.
NoScroll	Prevents the topic title from scrolling; overrides [HelpOptions]TitleScroll=Yes. See §17.7.3.2 <a href="#">Deciding whether to scroll titles</a> on page 298
NoTitle	Prevents the topic title from being displayed; when applied to a pop-up topic, prevents the topic from appearing in full-text search.

**Table 17-2 Effects of format properties on topics and hotspots (continued)**

Property	Effect
NoXScroll	Overrides [HelpOptions]ExtendHelpNoScroll=No for pop-ups. See §17.7.3.2 <a href="#">Deciding whether to scroll titles</a> on page 298.
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 §17.6.2 <a href="#">Displaying graphics in pop-ups for WinHelp</a> on page 293.
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. <b>DITA2Go</b> creates the “#” footnote for you. Omit this property if the topic is accessed only from cross references or hypertext links, and from the Contents.
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 §17.7.1 <a href="#">Creating WinHelp topics</a> on page 294.
Scroll	Causes the topic title to scroll; overrides [HelpOptions]TitleScroll=No. See §17.7.3.2 <a href="#">Deciding whether to scroll titles</a> on page 298.
Slide	Makes the topic a sliding topic. See §17.7.1 <a href="#">Creating WinHelp topics</a> on page 294.
SpKey	Produces a keyword footnote designated with a different letter (neither “A” nor “K”) See §17.10.3 <a href="#">Adding related-topic keywords with formats</a> on page 304.
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.
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 §17.7.3.1 <a href="#">Categorizing titles</a> on page 297.
Topic	Ends any prior topics and starts a new topic. <b>DITA2Go</b> creates the WinHelp topic start coding, a page break, and a title (\$) footnote, using the tagged text as the title. See §17.7.1 <a href="#">Creating WinHelp topics</a> on page 294.
Uline	Underlines a hotspot without turning it green.
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 format=Window; <b>DITA2Go</b> creates the “>” footnote. See §17.8.6 <a href="#">Specifying jumps to secondary windows in WinHelp</a> on page 302.
XScroll	Overrides [HelpOptions]ExtendHelpNoScroll=Yes for pop-ups. See §17.7.3.2 <a href="#">Deciding whether to scroll titles</a> on page 298.

## 17.7.3 Configuring topic titles for WinHelp

*In this section:*

§17.7.3.1 [Categorizing titles](#) on page 297

§17.7.3.2 [Deciding whether to scroll titles](#) on page 298

§17.7.3.3 [Fine-tuning title appearance](#) on page 298

### 17.7.3.1 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
```

### 17.7.3.2 Deciding whether to scroll titles

*Keep title from scrolling* 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:

```
[HelpOptions]
; TitleScroll = Yes (title para scrolls with text),
; or No (fixed at top)
TitleScroll=No
```

*Add to no-scroll region* You can extend the no-scroll region to include text that follows the title by setting the additional paragraphs to have the *Keep With Next* property, and by specifying the following:

```
[HelpOptions]
; ExtendHelpNoScroll = No (default),
; or Yes (allow Keep With Next paras)
ExtendHelpNoScroll=Yes
```

*Override no-scroll extension* To keep the ExtendHelpNoScroll=Yes setting but override it for particular paragraph formats, for each such format specify the following:

```
HelpStyles]
YourFormat=NoXScroll
```

**Note:** If a paragraph format is already listed under [HelpStyles], just add the NoXScroll property to its current list of properties; do not repeat the format name.

*Scroll title* To make titles scroll with the text, specify the following:

```
[HelpOptions]
TitleScroll=Yes
```

*Override scrolling* You can still create a no-scroll region for such topics with this setting:

```
[HelpOptions]
ExtendHelpNoScroll=Yes
```

*Pop-ups cannot be scrolled* Pop-up topics do not have scroll bars, so you might have to override the TitleScroll and ExtendHelpNoScroll settings. For each pop-up-topic paragraph format, add properties Scroll and NoXScroll. For example:

```
[HelpStyles]
PopTopic=Topic Scroll NoXScroll
```

### 17.7.3.3 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, **DITA2Go** 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
```



## 17.8 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.

*In this section:*

- §17.8.1 [Configuring pop-up topics](#) on page 299
- §17.8.2 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 299
- §17.8.3 [Using cross references for jumps and pop-ups](#) on page 300
- §17.8.4 [Using hypertext links for jumps and pop-ups](#) on page 301
- §17.8.5 [Disallowing hypertext links for jumps and pop-ups](#) on page 301
- §17.8.6 [Specifying jumps to secondary windows in WinHelp](#) on page 302
- §17.8.7 [Specifying jumps to external files](#) on page 302

*See also:*

- §16.7 [Jumping to secondary windows in Help systems](#) on page 262
- §16.8 [Creating pop-up topics for Help systems](#) on page 263
- §17.4.3 [Specifying cross-reference jump destinations](#) on page 289
- §17.5.3 [Converting table rows to topics and table cells to pop-ups](#) on page 292
- §17.6.2 [Displaying graphics in pop-ups for WinHelp](#) on page 293

### 17.8.1 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
```

*See also:*

- §17.7.1 [Creating WinHelp topics](#) on page 294
- §17.8.4.2 [Using alert or alerttitle markers for embedded pop-ups](#) on page 301

### 17.8.2 Creating hotspots for jumps and pop-ups in WinHelp

To create a hotspot, in DITA XML specify a dedicated `@outputclass` for 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 DITA XML.

*In this section:*

- §17.8.2.1 [Configuring jump vs. pop-up hotspots](#) on page 300
- §17.8.2.2 [Controlling hotspot appearance](#) on page 300
- §17.8.2.3 [Embedding hotspots in graphics for WinHelp](#) on page 300

See also:

§16.8.1 [Understanding pop-up hotspots, links, and topics](#) on page 263

### 17.8.2.1 Configuring jump vs. pop-up hotspots

<i>Pop-up hotspot</i>	For a pop-up hotspot, assign property PopOver to the hotspot character format: <pre>[HelpStyles] Hotspotfmt=PopOver</pre> Specify PopOver as the first property to the right of the equals sign. Additional properties can follow PopOver; see §17.7.2 <a href="#">Assigning properties to formats for topics and hotspots</a> on page 295.
<i>Jump hotspot</i>	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.
<i>Cross reference as a hotspot</i>	When you use a cross reference for a hotspot, make sure @outputclass is applied to the reference string in the cross-reference definition. If another inline element intervenes, 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 PI markers within the hotspot area.

### 17.8.2.2 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.

### 17.8.2.3 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 DITA document as **HyperAnchor** PI markers (see §17.8.4 [Using hypertext links for jumps and pop-ups](#) on page 301), and specify whether each hotspot should be a pop-up or a jump.

After you create a .shg file, to make **DITA2Go** reference the hypergraphic, specify the following configuration settings:

```
[Graphics]
FilePaths=None
FileNames=Map

[GraphFiles]
myold.bmp=mynew.shg
```

## 17.8.3 Using cross references for jumps and pop-ups

<i>Cross references as jumps</i>	By default, <b>DITA2Go</b> converts all DITA 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 §17.8.2.1 <a href="#">Configuring jump vs. pop-up hotspots</a> on page 300. 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>

## 17.8.4 Using hypertext links for jumps and pop-ups

By default, **DITA2Go** converts all DITA hypertext links into jumps for WinHelp, including interfile jumps where needed. You can insert hypertext PI markers in your DITA document to create additional jumps and pop-ups.

*In this section:*

§17.8.4.1 [Using HyperPopup and HyperJump PI markers](#) on page 301

§17.8.4.2 [Using alert or alerttitle markers for embedded pop-ups](#) on page 301

### 17.8.4.1 Using HyperPopup and HyperJump PI markers

Use a DITA **HyperPopup** or **HyperJump** PI marker to identify a hotspot, and type the reference string as the PI 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 §17.8.2 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 299.

**Note:** Do not place any other PI markers within the hotspot area.

### 17.8.4.2 Using alert or alerttitle markers for embedded pop-ups

Instead of creating a pop-up topic, you can use a **HyperAlert** or **HyperAlertTitle** PI 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 §17.8.4.1 [Using HyperPopup and HyperJump PI markers](#) on page 301. Type the marker text in the PI marker 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.

## 17.8.5 Disallowing hypertext links for jumps and pop-ups

If you do not want **DITA2Go** 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.

## 17.8.6 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 .hpl 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.

## 17.8.7 Specifying jumps to external files

DITA2Go produces links to external sources from **HyperLink** and **HyperFile** PI markers:

When a **HyperFile** link specifies an absolute path (which must start with a drive letter), DITA2Go removes any "file:/" URL prefix to the path, which is not needed in RTF. For example:

```
HyperFile="file:///g:/omnisys/ug/out/dita2go.pdf"
```

becomes:

```
!EF('g:/omnisys/ug/out/dita2go.pdf')
```

**HyperLink** and **HyperFile** are implemented in WinHelp with the !EF macro; see §17.9.1 [Using a hypertext marker to invoke a macro](#) on page 302.

## 17.9 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 an inline element with a special @outputclass to the hotspot text in DITA XML, the same way you would for a jump or a pop-up; see §17.8.2 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 299.

*In this section:*

§17.9.1 [Using a hypertext marker to invoke a macro](#) on page 302

§17.9.2 [Assigning a hotspot property to invoke a macro](#) on page 303

### 17.9.1 Using a hypertext marker to invoke a macro

The simplest way to create a jump to an external file is to insert a DITA **HyperLink** PI marker where you want the link. For example, to run an .avi video clip:

```
HyperLink="yourname.avi"
```

Indicate the hotspot area with a character format that includes the marker; see §17.8.2 [Creating hotspots for jumps and pop-ups in WinHelp](#) on page 299. DITA2Go converts the marker text to produce the required WinHelp macro invocation:

```
!EF('yourname.avi')
```

Clicking the hotspot should display the .avi file.

*See also:*

§17.8.7 [Specifying jumps to external files](#) on page 302

## 17.9.2 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')
```

**DITA2Go** 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/)
```

**DITA2Go** supplies the leading “!” for the macro invocation.

## 17.10 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 PI markers for both ALink jumps and ALink-list targets.

**DITA2Go** constructs KLinks for WinHelp 4 automatically, from DITA index entries. You can add other “K” footnotes that do not appear in the **DITA2Go**-generated index, either with formats or with PI markers.

*Advantages of  
markers*

Using PI markers for related-topic links has some advantages:

- The same markers work for ALinks and KLinks in all other **DITA2Go**-generated Help systems that support related-topic links.
- **DITA2Go** creates the WinHelp ALink or KLink macros for you when you use PI markers.

*In this section:*

§17.10.1 [Understanding KLink limitations](#) on page 303

§17.10.2 [Adding ALinks and KLinks with PI markers](#) on page 304

§17.10.3 [Adding related-topic keywords with formats](#) on page 304

§17.10.4 [Inserting WinHelp macros for ALink jumps](#) on page 305

*See also:*

§16.6 [Providing related-topic links for Help systems](#) on page 258

### 17.10.1 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.

## 17.10.2 Adding ALinks and KLinks with PI markers

To create ALinks and KLinks in WinHelp with PI markers, use the methods described in the following sections:

§16.6.4.1 [Adding related-topic link keywords via PI markers](#) on page 260

§16.6.5 [Adding ALink and KLink jumps in DITA XML](#) on page 261

## 17.10.3 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 §16.6.2 <a href="#">Understanding how ALinks work</a> on page 259  |
| Key   | a “K” footnote, for a KLink; see §16.6.3 <a href="#">Understanding how KLinks work</a> on page 259  |
| 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 an inline element with a special @outputclass to the relevant text, and assign a keyword property to the resulting 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 an inline element with a special @outputclass to the footnote text.
3. Assign properties AKey (for example) and Delete to the resulting 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
```

### 17.10.4 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,...)}
```

## 17.11 Configuring index entries for WinHelp

**DITA2Go** converts DITA index entries into WinHelp “K” footnotes. You can add to the WinHelp index other text that is not part of the DITA index, by assigning properties to formats or by inserting markers.

*In this section:*

§17.11.1 [Designating index level separators](#) on page 305

§17.11.2 [Eliminating duplicate keywords](#) on page 305

§17.11.3 [Keeping or discarding “See also” entries](#) on page 306

*See also:*

§16.5 [Configuring index entries for Help systems](#) on page 251

### 17.11.1 Designating index level separators

To have **DITA2Go** treat commas in DITA 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.

### 17.11.2 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.

### 17.11.3 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
```

## 17.12 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:*

- §17.12.1 [Naming and configuring Help files and titles](#) on page 306
- §17.12.2 [Specifying heading formats and levels for contents](#) on page 307
- §17.12.3 [Assembling WinHelp contents from the command line](#) on page 309

*See also:*

- §16.3.2 [Modifying contents or index production for WinHelp](#) on page 249
- §16.4 [Configuring contents entries for Help systems](#) on page 250

### 17.12.1 Naming and configuring Help files and titles

DITA2Go 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 .hbj (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; **DITA2Go** creates the .cnt file for you.

If you have more than one topic file in your Help project, set CntType=Body; **DITA2Go** 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, **DITA2Go** 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, **DITA2Go** uses the help file title (in the .hbj 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

```

## 17.12.2 Specifying heading formats and levels for contents

*In this section:*

§17.12.2.1 [Understanding WinHelp contents level numbers](#) on page 307

§17.12.2.2 [Listing topics for contents with and without subheadings](#) on page 308

§17.12.2.3 [Using different names in contents for heading and topic](#) on page 308

§17.12.2.4 [Renaming or eliminating the contents “Overview” topic](#) on page 309

§17.12.2.5 [Referencing multiple help files from contents](#) on page 309

§17.12.2.6 [Displaying contents targets in the main window](#) on page 309

### 17.12.2.1 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. **DITA2Go** produces these lines for each format value in [HelpStyles] that starts with Topic and includes Contents.

**DITA2Go** 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
```

### 17.12.2.2 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. <b>DITA2Go</b> 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. <b>DITA2Go</b> 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, <b>DITA2Go</b> 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.

### 17.12.2.3 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
```

#### 17.12.2.4 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 **DITA2Go** 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 §17.13 [Creating browse sequences](#) on page 310.

#### 17.12.2.5 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
```

#### 17.12.2.6 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
```

### 17.12.3 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 `hcv` line.

See §45 [Converting via DCL](#) on page 809.

## 17.13 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:*

§17.13.1 [Setting up an automatic browse sequence](#) on page 310

§17.13.2 [Specifying browse numbers](#) on page 310

§17.13.3 [Setting up multi-file browse sequences](#) on page 311

§17.13.4 [Setting up branching browse sequences](#) on page 311

### 17.13.1 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 .hpl 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 .hpl, 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.

### 17.13.2 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 PI marker (see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767).

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
```

### 17.13.3 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 PI marker in the DITA file (see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767)
- in a configuration file specifically for each .ditamap file (see §42.1 [Using a different configuration for selected files](#) on page 765).

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 d2winhelp.ini file.

### 17.13.4 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 §17.13.3 [Setting up multi-file browse sequences](#) on page 311):

```
[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
```



# 18 Generating Microsoft HTML Help

---

This section addresses issues that are specific to creating Microsoft HTML Help. HTML settings described in section 22 and sections 27 through 43 apply also. Topics include:

- §18.1 [Understanding how DITA2Go produces HTML Help](#) on page 313
- §18.2 [Understanding why Unicode is not the answer](#) on page 314
- §18.3 [Setting up an HTML Help project](#) on page 315
- §18.4 [Customizing HTML Help display features](#) on page 319
- §18.5 [Creating pop-ups for HTML Help](#) on page 322
- §18.6 [Creating links and hypertext jumps in HTML Help](#) on page 323
- §18.7 [Creating related-topic links for HTML Help](#) on page 325
- §18.8 [Using secondary windows in HTML Help](#) on page 332
- §18.9 [Generating contents and index for HTML Help](#) on page 334
- §18.10 [Providing full-text search \(FTS\) for HTML Help](#) on page 339
- §18.11 [Setting up CSH for HTML Help](#) on page 340
- §18.12 [Generating HTML Help in non-Western languages](#) on page 344
- §18.13 [Compiling and testing HTML Help](#) on page 346
- §18.14 [Mapping and merging CHM files](#) on page 348

*See also:*

- §16 [Producing on-line Help](#) on page 243

To determine which configuration settings will produce the appearance and functionality you want, see also:

- §22 [Converting to HTML/XHTML](#) on page 429
- §27 [Splitting and extracting files](#) on page 523
- §30 [Mapping text formats to HTML/XML](#) on page 565
- §31 [Setting up CSS for HTML](#) on page 591
- §32 [Including graphics in HTML](#) on page 611
- §33 [Converting tables to HTML](#) on page 625

For more information about HTML Help, see the Microsoft HTML Help home page, accessible through the Microsoft Library:

<http://msdn.microsoft.com/library>

## 18.1 Understanding how DITA2Go produces HTML Help

Microsoft HTML Help is specialized for use with Microsoft HTML Help Workshop, which is used to compile the HTML files **DITA2Go** 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 **DITA2Go** cannot address.

To produce HTML Help, **DITA2Go** does the following:

- creates an HTML Help project file
- generates HTML topic files from your DITA document
- optionally runs HTML Help Workshop to compile the HTML Help project.



<i>Initial project file</i>	When you create a <b>DITA2Go</b> HTML Help project, <b>DITA2Go</b> automatically generates a starting Microsoft HTML Help project file. This file is named for your DITA document, with extension <code>.hhp</code> , and placed in the project directory. For example, if you are converting <code>MyDoc.map</code> , <b>DITA2Go</b> 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, <b>DITA2Go</b> does not touch the HTML Help project file again, though you can tell <b>DITA2Go</b> to regenerate it each time you run a conversion; see §18.3.8 <a href="#">Regenerating the HTML Help project file</a> on page 318.
<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 DITA 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 <b>DITA2Go</b> 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 <b>Properties</b> , and click <b>Unblock</b> , then click <b>Apply</b> , then <b>OK</b> . After that, when you double-click the CHM, it opens fine. If you select <b>Properties</b> again, the area where the <b>Unblock</b> 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 <b>DITA2Go</b> to generate HTML Help, choose configuration options that do not produce <code>&lt;object&gt;</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); and adding navigation (see §29 <a href="#">Providing navigation in HTML</a> on page 555) to the top and bottom of each output page. Also see §18.6.2 <a href="#">Specifying href link syntax for HTML Help</a> on page 324.

## 18.2 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. **DITA2Go** goes to considerable lengths to convert from Unicode to code page for HTML Help. It is not trivial; for Asian languages, **DITA2Go** 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:*

§18.12 [Generating HTML Help in non-Western languages](#) on page 344

§30.4 [Assigning properties to text formats](#) on page 570

## 18.3 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 §18.12 [Generating HTML Help in non-Western languages](#) on page 344.

*In this section:*

§18.3.1 [Creating an HTML Help project](#) on page 315

§18.3.2 [Deciding where to locate configuration settings](#) on page 316

§18.3.3 [Organizing source files for HTML Help](#) on page 316

§18.3.4 [Specifying a project title for HTML Help](#) on page 316

§18.3.5 [Deciding whether to compile HTML Help](#) on page 317

§18.3.6 [Naming project and compiled files for HTML Help](#) on page 317

§18.3.7 [Specifying a starting topic file for HTML Help](#) on page 317

§18.3.8 [Regenerating the HTML Help project file](#) on page 318

§18.3.9 [Locating graphics files for HTML Help](#) on page 318

### 18.3.1 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 DITA document is located. Optionally, create a subdirectory for graphics files.
2. Copy configuration file `_d2htmlhelp.ini` from your **DITA2Go** config directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:
3. Use a text editor to edit `_d2htmlhelp.ini` (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

4. **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 §18.3.2 [Deciding where to locate configuration settings](#) on page 316.

## 18.3.2 Deciding where to locate configuration settings

When you set up an HTML Help project, if configuration file `_d2htmlhelp.ini` is not already present in the project directory, you must copy this file from your **DITA2Go** `config\local` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29).

*Which  
configuration file?*

To configure HTML Help 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	<code>_d2htmlhelp.ini</code>	Current project directory
All HTML Help projects	<code>local_d2htmlhelp_config.ini</code>	<code>%omsyshome%\d2g\local\config\</code>

See §39.4 [Deciding which configuration file to edit](#) on page 734.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §22 [Converting to HTML/XHTML](#) on page 429
- §27 [Splitting and extracting files](#) on page 523
- §30 [Mapping text formats to HTML/XML](#) on page 565
- §32 [Including graphics in HTML](#) on page 611
- §33 [Converting tables to HTML](#) on page 625

## 18.3.3 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 DITA 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, **DITA2Go** can copy the graphics files to the project directory (or a subdirectory) before beginning the conversion. See §18.3.9 [Locating graphics files for HTML Help](#) on page 318.

## 18.3.4 Specifying a project title for HTML Help

The title of your HTML Help project appears in the title bar of the HTML Help viewer.

To specify a title for your help project:

```
[MSHtmlHelpOptions]
; HelpFileTitle = title to put in project file;
; default is filename or bookname
HelpFileTitle=Title of my project
```

If your CHM file will be used in locales other than US English, also see §18.12 [Generating HTML Help in non-Western languages](#) on page 344.

### 18.3.5 Deciding whether to compile HTML Help

The HTML Help compiler does not support Unicode, and instead uses code-page mappings. For compiled HTML Help, **DITA2Go** 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 **DITA2Go** *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 27

§18.12 [Generating HTML Help in non-Western languages](#) on page 344

§18.13 [Compiling and testing HTML Help](#) on page 346.

### 18.3.6 Naming project and compiled files for HTML Help

By default, **DITA2Go** uses the name of your DITA document for both the project file (`.hhp`) and the compiled file (`.chm`).

*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 DITA document.

**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 §18.14 [Mapping and merging CHM files](#) on page 348.

### 18.3.7 Specifying a starting topic file for HTML Help

By default, **DITA2Go** puts the name of your DITA document in the `.hhp` as the name of the first page. If the first page really should be the first *split* file (see §27 [Splitting and extracting files](#) on page 523), 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 `aa123456.htm`):

```
[OPTIONS]
Default topic=realfilename.htm
```

Also specify the starting topic in configuration file d2htmlhelp.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.

### 18.3.8 Regenerating the HTML Help project file

When you use **DITA2Go** to generate HTML Help, **DITA2Go** writes an .hhp project file during set-up, and rewrites it later only under certain conditions.

To specify whether **DITA2Go** 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, <b>DITA2Go</b> overwrites it.
No	<b>DITA2Go</b> does not overwrite the .hhp file.
(none)	If the configuration chain contains no WriteHelpProjectFile setting at all, <b>DITA2Go</b> writes an .hhp file, but <i>only if the .hhp file is not already present</i> .

**DITA2Go** closes the .hhp file after writing it; so, if you had the .hhp file open in HTML Help Workshop when **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** configuration file to preserve them; see §18.8.1 [Defining secondary windows for HTML Help](#) on page 333.

### 18.3.9 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. **DITA2Go** can do this for you.

To tell **DITA2Go** to fetch your referenced graphics:

```
[Automation]
WrapAndShip = Yes
CopyOriginalGraphics = Yes
```

When `CopyOriginalGraphics=Yes`, **DITA2Go** follows the file paths in your DITA source to find the graphics files to copy.

To tell **DITA2Go** 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 §44.3 [Understanding path values for deliverables](#) on page 788). 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, **DITA2Go** converts them to forward slashes before inserting the references in your HTML output. If you specify `CopyOriginalGraphics=Yes`, **DITA2Go** copies graphics files to the directory specified by `GraphPath`, after generating HTML files.

*See also:*

§18.13 [Compiling and testing HTML Help](#) on page 346

§32.1 [Locating graphics files for HTML](#) on page 611

§44.7 [Placing graphics files for distribution](#) on page 796

## 18.4 Customizing HTML Help display features

*In this section:*

§18.4.1 [Using CSS and font tags with HTML Help](#) on page 319

§18.4.2 [Adding tabs and toolbar buttons to HTML Help](#) on page 319

§18.4.3 [Adding expandable sections to HTML Help](#) on page 321

### 18.4.1 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 §18.4.2 [Adding tabs and toolbar buttons to HTML Help](#) on page 319), CSS font sizes must be in relative units: `em`, `ex`, or `%`. For best practice, use `em`. By default, **DITA2Go** generates CSS entries using absolute `pt` units for font size and line height. To change the units, see §31.8.2 [Specifying CSS size values and units of measurement](#) on page 607.

*See also:*

§31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593

### 18.4.2 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 §18.9.6 [Providing mid-topic contents links in HTML Help](#) on page 337. Also, a binary TOC is not compatible with merged CHM files; see §18.14.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 351.

To enable additional HTML Help tabs and toolbar buttons:

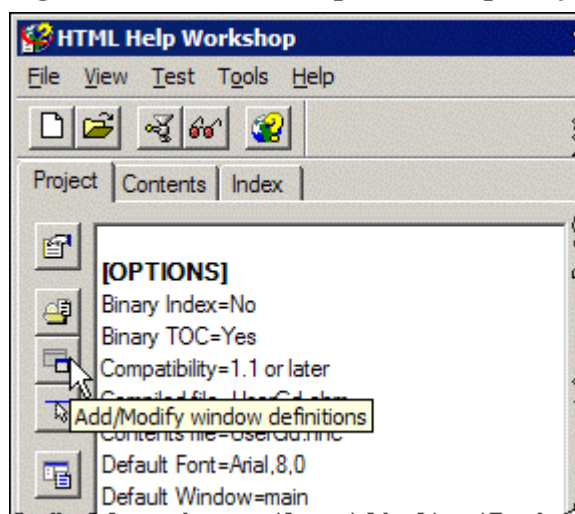
1. Set the following option in your project configuration file, to avoid overwriting the changes you are about to make to the HTML Help .hhp project file:

```
[MShtmlHelpOptions]
WriteHelpProjectFile=No
```

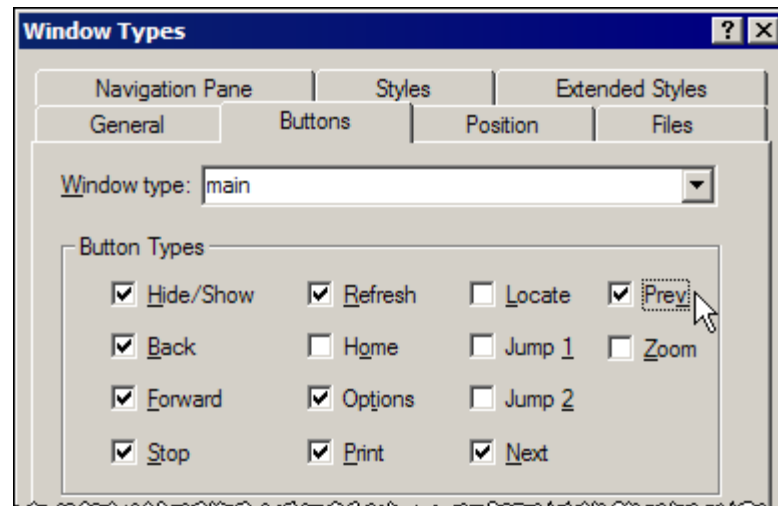
See §18.3.8 [Regenerating the HTML Help project file](#) on page 318

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 18-1 HTML Help Workshop Project tab**



5. On the **Project** tab toolbar, click **Add/Modify window definitions**, as shown in [Figure 18-1](#).
6. If the *Add a New Window Type* dialog opens, type **main**, and click **OK**; the *Window Types* dialog opens, as shown in [Figure 18-2](#) on page 321.
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 18-2 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 §18.4.1 [Using CSS and font tags with HTML Help](#) on page 319.

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**.

### 18.4.3 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 **DITA2Go** 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 DITA XML. Or, you could use a button (see §18.7.6 [Creating buttons for other types of related-topic links](#) on page 332).

## 18.5 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:*

§18.5.1 [Using HTML Help for pop-ups](#) on page 322


§18.5.2 [Using KeyHelp for pop-ups](#) on page 322

§18.5.3 [Using WinHelp for pop-ups](#) on page 323

### 18.5.1 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 PI **alert** marker, 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.

### 18.5.2 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](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 **DITA2Go** macros (see §37 [Working with macros](#) on page 679) 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
```

### 18.5.3 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 §17.8 [Creating jumps and pop-ups for WinHelp](#) on page 299.

## 18.6 Creating links and hypertext jumps in HTML Help

*In this section:*

§18.6.1 [Creating hypertext jumps to other CHM files](#) on page 323

§18.6.2 [Specifying href link syntax for HTML Help](#) on page 324

§18.6.3 [Linking to external files from compiled HTML Help](#) on page 325

*See also:*

§18.7 [Creating related-topic links for HTML Help](#) on page 325

### 18.6.1 Creating hypertext jumps to other CHM files

If you are using hypertext links to link to HTML files created from a different DITA document and compiled into a different .chm, you must specify how the CHM files are mapped; see §18.14.1 [Interlinking multiple CHM files](#) on page 348. Then **DITA2Go** 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 a link in your document:

```
<xref href="someother.chm" scope="external"></xref>
```

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 link content; for example, to get to the “a” anchor inside *letters.htm* in *someother.chm*:

```
<xref href="someother.chm::/letters.htm#a" scope="external"></xref>
```

*Specific topic* To jump to a specific topic in *someother.chm*, when the topic comes from a DITA file that is being split (so you do not know the .htm file name ahead of time), insert a regular DITA cross reference. Run the **DITA2Go** 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 §18.6.3 [Linking to external files from compiled HTML Help](#) on page 325.

## 18.6.2 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, **DITA2Go** 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 §18.6.3 [Linking to external files from compiled HTML Help](#) on page 325.

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 §18.14.1 [Interlinking multiple CHM files](#) on page 348.)

*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:dita2go.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 **DITA2Go** 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.

### 18.6.3 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 **DITA2Go** macro to provide the JavaScript; see §37 [Working with macros](#) on page 679.

*Linking to Web sites*

For external links on the Web, you can use 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
```

## 18.7 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:*

- §18.7.1 [Adding ALink keywords for HTML Help](#) on page 325
- §18.7.2 [Adding ALink and KLink jumps for HTML Help](#) on page 326
- §18.7.3 [Configuring ALink and KLink jumps for HTML Help](#) on page 326
- §18.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 328
- §18.7.5 [Using the same format for ALink keywords and jumps](#) on page 329
- §18.7.6 [Creating buttons for other types of related-topic links](#) on page 332

### 18.7.1 Adding ALink keywords for HTML Help

You can insert ALink keywords for HTML Help either with PI markers or with paragraph formats; see §16.6.4 [Adding related-topic link keywords in DITA XML](#) on page 260.

ALink keywords should be single terms; use spaces or other punctuation in ALink keywords at your own risk.

## 18.7.2 Adding ALink and KLink jumps for HTML Help

You can use PI markers or paragraph formats for ALink and KLink jumps; however, **DITA2Go** provides support only for PI markers. In HTML Help, a related-topic jump is implemented with an <object> linked to the HTML Help OLE control:

- |                |  |
|----------------|--|
| <b>Markers</b> | If you use PI markers for ALink or KLink jumps, <b>DITA2Go</b> provides the <object> macro code automatically; see §18.7.3 <a href="#">Configuring ALink and KLink jumps for HTML Help</a> on page 326. Put the PI markers in paragraphs by themselves, located where you want the jump to appear in output. |
| <b>Formats</b> | If you use paragraph formats for ALink or KLink jumps, you have to provide macro code for the ALink object; see §18.7.4 <a href="#">Rolling your own macros for ALink jumps in HTML Help</a> on page 328.  |

See also §16.6.5 [Adding ALink and KLink jumps in DITA XML](#) on page 261.

**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.

## 18.7.3 Configuring ALink and KLink jumps for HTML Help

When you use PI markers for ALink or KLink jumps (see §16.6.5 [Adding ALink and KLink jumps in DITA XML](#) on page 261), you can specify values in the configuration file for several properties of the resulting <object>s that **DITA2Go** 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
```

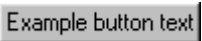


Table 18-1 on page 327 shows the properties you can configure, the values you can specify for each property, and the effect of each value.

<i>Prefix the keyword</i>	The base keyword for each property starts with <code>Link</code> ; for example, <code>LinkFlags</code> . You must add a prefix to the base keyword to create a valid setting: <ul style="list-style-type: none"> <li>To specify an <code>ALink</code> property, prefix the base name with <code>A</code>; for example, <code>ALinkFlags=, ,1</code>.</li> <li>To specify a <code>KLink</code> property, prefix the base name with <code>K</code>; for example, <code>KLinkButtonWidth=50</code>.</li> </ul>
<i>All jumps</i>	Properties you specify this way apply to all <code>ALink</code> or <code>KLink</code> jumps in your document.
<i>Selected jumps</i>	To configure properties for an individual <code>ALink</code> or <code>KLink</code> jump, insert a <b>Link*</b> PI marker just before the jump marker. The <b>Link*</b> 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 <code>ALink</code> jump, in your DITA document, just before the <b>ALink</b> jump marker, insert a <b>LinkText</b> PI marker whose content is the alternate label.
<i>Specify all relevant properties</i>	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 <code>&lt;object&gt;</code> . For example, for an <code>ALink</code> jump the default object <b>DITA2Go</b> 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 18-1 ALink and KLink jump properties for HTML Help**

Property keyword*	Value	Effect
<code>LinkType</code>	Button <i>(default)</i>	Link is a button with <code>LinkButtonWidth</code> , <code>LinkButtonHeight</code> , and <code>LinkButtonText</code> attributes: 
	Chiclet	Link is a small button with no label: 
	Graphic	Link is a bitmap image, specified by <code>LinkButtonGraphic</code>
	Icon	Link is an icon, specified by <code>LinkButtonIcon</code>
	Shortcut	Link is a button with a shortcut icon: 
	Text	Link is text, with <code>LinkText</code> and <code>LinkTextFont</code> attributes
<code>LinkFlags</code>	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.

\* The property keyword must be prefixed with "A" for an `ALink` property, or "K" for a `KLink` property.

**Table 18-1 ALink and KLink jump properties for HTML Help (continued)**

Property keyword*	Value	Effect
LinkEmptyTopic	<i>noitems.htm</i>  <i>(none)</i>	Name of .htm topic file to show if no targets are found  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
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 §18.5 <a href="#">Creating pop-ups for HTML Help</a> on page 322)

\* The property keyword must be prefixed with "A" for an ALink property, or "K" for a KLink property.

## 18.7.4 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="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: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 §18.7.5 [Using the same format for ALink keywords and jumps](#) on page 329

## 18.7.5 Using the same format for ALink keywords and jumps

You can use the same paragraph format both for ALink keywords and to produce a button for an ALink jump in HTML Help. Use configuration settings and **DITA2Go** 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. The only restriction is that ALink keyword paragraphs must precede the location in each topic where you want the ALink jump to appear.

*In this section:*

§18.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 329

§18.7.5.2 [Initializing the ALink keyword list counter](#) on page 330

§18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330

§18.7.5.4 [Positioning the ALink button in each HTML Help topic](#) on page 331

§18.7.5.5 [Including multiple ALink keywords in a paragraph or marker](#) on page 331

### 18.7.5.1 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
```

<i>ALink property</i>	The <i>ALink</i> property specifies that the content of each <i>ALinkTarget</i> paragraph is to be used for the <i>ALink Name</i> property of an HTML Help ALink object ( <i>not</i> the button object, which you will construct with macros); see §16.6.4.2 <a href="#">Adding related-topic keywords via format properties</a> on page 260.
<i>Raw property</i>	The <i>Raw</i> property suppresses any HTML tags that would otherwise be generated; see §30.2.4 <a href="#">Stripping paragraph properties</a> on page 568.
<i>CodeStore property</i>	The <i>CodeStore</i> property causes the content of the <i>ALinkTarget</i> paragraph to be stored in macro variable <code>\$\$ALinkTarget</code> . (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 §37.3.1 <a href="#">Creating and invoking macro variables</a> on page 687.) The <i>CodeStore</i> property also removes the paragraph from text output; see §37.3.5.2 <a href="#">Inserting code with the CodeStore property</a> on page 693.
<i>CodeAfter property</i>	The <i>CodeAfter</i> property provides the means to do something further with macro variable <code>\$\$ALinkTarget</code> , which now contains an ALink keyword, plucked from the <i>ALinkTarget</i> paragraph:
	<pre>[ParaStyleCodeAfter] ALinkTarget=&lt;\$\$Nkeys++&gt;&lt;\$\$ALinkKeys[\$\$Nkeys]=\$\$ALinkTarget&gt;</pre>
<i>Store paragraph content in a list variable</i>	<p>The <code>[ParaStyleCodeAfter]</code> code does the following:</p> <ul style="list-style-type: none"> <li>• Increments a counter, <i>Nkeys</i> (which will be initialized to zero before each topic).</li> <li>• Uses <i>Nkeys</i> to index a list variable, <code>\$\$ALinkKeys</code> (see §37.4 <a href="#">Using multiple-value list variables</a> on page 695).</li> </ul>

- Stores the content of macro variable `$$ALinkTarget` in the `Nkeys` slot in list variable `$$ALinkKeys`.

As **DITA2Go** processes DITA 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 §18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330.

See also:

§18.7.5.2 [Initializing the ALink keyword list counter](#) on page 330

§18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330

§18.7.5.4 [Positioning the ALink button in each HTML Help topic](#) on page 331

### 18.7.5.2 Initializing the ALink keyword list counter

To set the counter for *ALink* keyword list variable `$$ALinkKeys` to zero at the beginning of each DITA file:

```
[MacroVariables]
Nkeys=0
```

The `[ALinkButton]` macro (see §18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330) sets `Nkeys` back to zero again after finishing each button, to re-initialize `Nkeys` for the next topic.

See also:

§18.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 329

§18.7.5.2 [Initializing the ALink keyword list counter](#) on page 330

### 18.7.5.3 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 §18.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 329) 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:

`$$ALinkParamNum`    Keyword parameter counter (the *N* in "Item*N*")  
`$$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 "Item*N*" keyword parameters for the button object start with *N*=2, not *N*=1 (see §18.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 328).

`[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 §18.7.4 [Rolling your own macros for ALink jumps in HTML Help](#) on page 328:

```
[ALinkButtonStart]
<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: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 §18.7.5.5 [Including multiple ALink keywords in a paragraph or marker](#) on page 331.

*See also:*

§18.7.5.4 [Positioning the ALink button in each HTML Help topic](#) on page 331

#### 18.7.5.4 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 §27.6 [Inserting HTML code in split and extract files](#) on page 534.

*See also:*

§18.7.5.1 [Creating a list of ALink keywords from paragraphs](#) on page 329

§18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330

#### 18.7.5.5 Including multiple ALink keywords in a paragraph or marker

An enhanced version of the [ALinkButton] macro (see §18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330) 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 §18.7.5.3 [Building an ALink button object from an ALink keyword list](#) on page 330:

```
[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 §37.6.4.3 [Using loop structures](#) on page 705 for an explanation of loop controls \$\_repeat and \$\_while.

See §37.6.5 [Specifying substrings in expressions](#) on page 706 for an explanation of string operators contains, before, and after.

## 18.7.6 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 §18.9.8 [Customizing contents and index for HTML Help](#) on page 338 for special DITA2Go settings that provide a way to specify contents and index parameters.

## 18.8 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:*

§18.8.1 [Defining secondary windows for HTML Help](#) on page 333

§18.8.2 [Jumping from a topic to a secondary window](#) on page 333

§18.8.3 [Jumping from contents or index to a secondary window](#) on page 333

*See also:*

§16.7 [Jumping to secondary windows in Help systems](#) on page 262

## 18.8.1 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 **DITA2Go** to rewrite your .hhp file each time (see §18.3.8 [Regenerating the HTML Help project file](#) on page 318), 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= ....
```

## 18.8.2 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 §18.14.1 [Interlinking multiple CHM files](#) on page 348.

*See also:*

§16.7 [Jumping to secondary windows in Help systems](#) on page 262

§18.8.1 [Defining secondary windows for HTML Help](#) on page 333

## 18.8.3 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:*

§18.8.3.1 [Assigning a secondary window with a paragraph format](#) on page 333

§18.8.3.2 [Assigning a secondary window with a marker](#) on page 334

### 18.8.3.1 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** PI 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** PI marker or paragraph assigned the `Window` property, determines which secondary window will be used for jumps from the index.

See also:

§18.8.1 [Defining secondary windows for HTML Help](#) on page 333

§18.8.3.2 [Assigning a secondary window with a marker](#) on page 334

### 18.8.3.2 Assigning a secondary window with a marker

To use a PI marker to force a jump from contents or index to a secondary window, insert a PI marker of type **Window** in the target topic. Supply the name of the secondary window as the content of the **Window** PI marker.

A **Window** PI 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 §18.8.3.1 [Assigning a secondary window with a paragraph format](#) on page 333.

*Jumps from contents* If you place a **Window** PI 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 PI marker. **Window** PI 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** PI marker or paragraph assigned the `Window` property, determines which secondary window will be used for jumps from the index.

See also:

§18.8.1 [Defining secondary windows for HTML Help](#) on page 333

§18.8.3.1 [Assigning a secondary window with a paragraph format](#) on page 333

## 18.9 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:

§18.9.1 [Choosing how to generate HTML Help contents and index](#) on page 335

§18.9.2 [Choosing whether to generate binary contents or index](#) on page 335

§18.9.3 [Generating contents and index with HTML Help Workshop](#) on page 336

§18.9.4 [Generating contents and index with DITA2Go](#) on page 336

§18.9.5 [Configuring contents entries for HTML Help](#) on page 337

§18.9.6 [Providing mid-topic contents links in HTML Help](#) on page 337

§18.9.7 [Making the TOC track index links in HTML Help](#) on page 338

§18.9.8 [Customizing contents and index for HTML Help](#) on page 338

## 18.9.1 Choosing how to generate HTML Help contents and index

To specify whether contents, index, or both should be generated for HTML Help:

```
[MSHtmHelpOptions]
; 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 **DITA2Go** or by HTML Help Workshop, or by neither. This choice is governed by the following setting:

```
[MSHtmHelpOptions]
; RefFileType = HHW (for HH Workshop), Full,
; or None (do not generate).
RefFileType = Full
```

When `RefFileType=Full`, **DITA2Go** generates contents and index for HTML Help.

When `RefFileType=HHW`, **DITA2Go** 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.

*See also:*

§16.3.1 [Modifying contents or index production for HTML-based Help](#) on page 249

§18.9.3 [Generating contents and index with HTML Help Workshop](#) on page 336.

## 18.9.2 Choosing whether to generate binary contents or index

To produce binary TOC and index for HTML Help:

```
[MSHtmHelpOptions]
; 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 §18.3.8 [Regenerating the HTML Help project file](#) on page 318.

Alternatively, you can specify binary TOC or index generation directly in the `.hbj` file, under `[Options]`; see §18.4.2 [Adding tabs and toolbar buttons to HTML Help](#) on page 319.

There are trade-offs to generating binary navigation features for HTML Help. [Table 18-2](#) lists the pros and cons of specifying a binary TOC or a binary index.

**Table 18-2 Binary TOC/Index advantages and disadvantages for HTML Help**

Binary feature	Advantages and disadvantages
TOC	<p><b>Pros:</b> Supports browse via <b>Prev</b> and <b>Next</b> buttons in the HTML Help viewer; see §18.4.2 <a href="#">Adding tabs and toolbar buttons to HTML Help</a> on page 319.</p> <p>Supports no-link entries in the TOC; see §18.9.5 <a href="#">Configuring contents entries for HTML Help</a> on page 337.</p> <p>Allows the TOC to stay synchronized with topics selected via the index; see §18.9.7 <a href="#">Making the TOC track index links in HTML Help</a> on page 338.</p> <p><b>Cons:</b> Incompatible with merging .chm files at run time; see §18.14.5 <a href="#">Comparing HHW settings for stand-alone vs. merged CHMs</a> on page 351.</p> <p>Can cause problems with mid-topic TOC links; see §18.9.6 <a href="#">Providing mid-topic contents links in HTML Help</a> on page 337.</p>
Index	<p><b>Pros:</b> Supports merging .chm files at run time; see §18.14.5 <a href="#">Comparing HHW settings for stand-alone vs. merged CHMs</a> on page 351.</p> <p><b>Cons:</b> Prevents index customization; see §16.5.8 <a href="#">Customizing index sort order</a> on page 256.</p>

### 18.9.3 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, **DITA2Go** 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, **DITA2Go** embeds DITA index entries (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.

### 18.9.4 Generating contents and index with DITA2Go

If you specify `RefFileType=Full`, **DITA2Go** creates contents or index files or both. See §16.3.1 [Modifying contents or index production for HTML-based Help](#) on page 249.

When you use the .hhp project file to compile a CHM file, **DITA2Go** 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 **DITA2Go** 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.



## 18.9.5 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:

§16.4.2 [Including contents entries in HTML-based Help](#) on page 250.

§16.4.3 [Setting contents levels for HTML-based Help](#) on page 251.

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 §18.9.6 [Providing mid-topic contents links in HTML Help](#) on page 337.

*Split at each heading level*

The best way to ensure that each TOC link goes to an exact destination is to split your DITA 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, **DITA2Go** moves the whole hierarchy up one level. In the example in §18.9.4 [Generating contents and index with DITA2Go](#) on page 336, 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:*

§16.4 [Configuring contents entries for Help systems](#) on page 250

§18.9.6 [Providing mid-topic contents links in HTML Help](#) on page 337

## 18.9.6 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; **DITA2Go** 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.

## 18.9.7 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 §18.14.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 351.

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.

## 18.9.8 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 **DITA2Go** 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">
```

**DITA2Go** 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 **DITA2Go** 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:

§16.4 [Configuring contents entries for Help systems](#) on page 250

§16.5 [Configuring index entries for Help systems](#) on page 251

## 18.10 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 §18.13 [Compiling and testing HTML Help](#) on page 346. With default settings, you get FTS automatically when **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** 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 all DITA topic headings and all other headings at which you split DITA files to create HTML Help topics. For example:

```
[HTMLParaStyles]
Heading1 = Split Title Contents
Heading2 = Split Title Contents
```

See §27.5.2 [Specifying page titles for split or extract files](#) on page 531.

**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 §18.13.3 [Compiling in a different language](#) on page 347.

**Excluding a topic from FTS** **DITA2Go** 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.

DITA topicref attribute `@search` (with value `yes` or `no`) allows you to specify whether the referenced topic is to be included in full-text search. **DITA2Go** excludes from HTML Help full-text search topics that are referenced with `@search="no"`.

If your DITA maps do not include values for the topicref `@search` attribute, you can achieve the same effect with PI markers inserted in the topicrefs, and a ditaval file. For example:

```
<?dthtm Search="no" ?>
```

In this example the effect on **DITA2Go** output is the same as if you had included `@search="no"` in the topicref for such topics.

## 18.11 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:*

§18.11.1 [Inserting CSH destinations in your document](#) on page 341

§18.11.2 [Determining whether you need map and alias files](#) on page 341

- §18.11.3 [Specifying and generating a map file for CSH links](#) on page 342
- §18.11.4 [Creating an alias file for CSH links](#) on page 343
- §18.11.5 [Understanding alias-file entries](#) on page 343
- §18.11.6 [Producing a list of aliases and associated topic titles](#) on page 344

See also:

- §16.10.1 [Understanding how CSH works](#) on page 278
- [http://helpware.net/htmlhelp/how\\_to\\_context.htm](http://helpware.net/htmlhelp/how_to_context.htm).

## 18.11.1 Inserting CSH destinations in your document

To insert CSH destinations in your DITA document, use PI markers.

Markers provide the only way to insert mid-topic CSH destinations.

To provide a CSH destination with a marker:

1. Place a `TopicAlias` PI marker in the text of your document where you want to display context-sensitive help. Markers can be anywhere in the text; a good place is in the topic, after the root and before the title. Each PI marker must be within the material you want presented to the user.
2. Make the content of the PI marker a symbolic ID: a unique name with a prefix you specify in the configuration file; see §18.11.4 [Creating an alias file for CSH links](#) on page 343.

*Mid-topic destinations*

Even if you insert the PI 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 `TopicAlias` PI marker containing a dummy symbolic ID at the start of your DITA document. Also arrange for a dummy link to this destination; see §18.11.3 [Specifying and generating a map file for CSH links](#) on page 342.

See §18.11.5 [Understanding alias-file entries](#) on page 343.

## 18.11.2 Determining whether you need map and alias files

**Markers** If you use PI 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 PI marker text.
- alias file* Associates each symbolic ID in your document with the `.htm` file where the relevant PI marker is located.

When you use PI 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 DITA document to HTML Help,

**DITA2Go** uses the symbolic IDs to generate an alias file; see §18.11.4 [Creating an alias file for CSH links](#) on page 343. 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 §18.11.3 [Specifying and generating a map file for CSH links](#) on page 342.

### 18.11.3 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 **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** from rewriting the `.hhp` file. See §18.3.8 [Regenerating the HTML Help project file](#) on page 318.

#### *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 **DITA2Go** 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
```

**DITA2Go** 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.

### 18.11.4 Creating an alias file for CSH links

When you use PI markers for CSH destinations, by default **DITA2Go** generates an alias file for you, named after your document, with extension `.hha`; for example, `MyDoc.hha`. **DITA2Go** 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 `TopicAlias` PI 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 `TopicAlias` PI marker in your document that contains a name prefixed with `HIDC_` or `IDH_`. See §18.11.5 [Understanding alias-file entries](#) on page 343 for examples.

*No alias file* To prevent **DITA2Go** from creating an alias file:

```
[MShtmlHelpOptions]
; MakeAliasFile = Yes (default, make list of newlinks and files) or No
MakeAliasFile=No
```

When `MakeAliasFile=No`, **DITA2Go** does not generate an alias file. In that case, if you are using PI 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 `TopicAlias` PI markers that qualify, **DITA2Go** does not generate an alias file.

### 18.11.5 Understanding alias-file entries

By default **DITA2Go** 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 **DITA2Go** to generate alias-file entries of this form, specify the following option:

```
[MShtmlHelpOptions]
UseAliasAName=Yes
```

See §18.11.1 [Inserting CSH destinations in your document](#) on page 341.

*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`, **DITA2Go** 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 `TopicAlias` PI 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.

### 18.11.6 Producing a list of aliases and associated topic titles

To direct **DITA2Go** 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`, **DITA2Go** 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 DITA2Go project"
IDH_Export "Converting documents"
```

## 18.12 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:*

- §18.12.1 [Converting from Unicode to Windows code pages](#) on page 344
- §18.12.2 [Specifying locale and language for HTML Help](#) on page 345
- §18.12.3 [Preventing inclusion of Unicode numeric references](#) on page 346

*See also:*

- §16.5.8.3 [Specifying index sort type and locale](#) on page 257
- §18.3.5 [Deciding whether to compile HTML Help](#) on page 317
- §18.13.3 [Compiling in a different language](#) on page 347

### 18.12.1 Converting from Unicode to Windows code pages

**DITA2Go** 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.

**DITA2Go** will use these code-page DLLs to prepare your HTML Help output, depending on the locale you specify; see §18.12.2 [Specifying locale and language for HTML Help](#) on page 345.

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)

## 18.12.2 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
```

**DITA2Go** 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      §22.4.3 Specifying character encoding for HTML on page 434
XMLEncoding   §23.2.3 Specifying character encoding for generic XML on
                page 450
```

*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”. **DITA2Go** 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, **DITA2Go** maps all fixed spaces to the ideographic space, U+3000 (x81 x40), for code page 932.

### 18.12.3 Preventing inclusion of Unicode numeric references

As a partial workaround for the lack of Unicode support, by default **DITA2Go** 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 §22.4.3 [Specifying character encoding for HTML](#) on page 434.

## 18.13 Compiling and testing HTML Help

It is best to compile HTML Help in a directory different from your **DITA2Go** HTML Help project directory. You can have **DITA2Go** 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:*

§18.13.1 [Directing DITA2Go to run the HTML Help compiler](#) on page 346

§18.13.2 [Copying output files and compiling later](#) on page 347

§18.13.3 [Compiling in a different language](#) on page 347

§18.13.4 [Registering your HTML Help system for network use](#) on page 347

### 18.13.1 Directing DITA2Go to run the HTML Help compiler

When you specify the following options in the configuration file, **DITA2Go** 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 **DITA2Go** finishes the conversion, compile directly from HTML Help Workshop. Otherwise you might encounter memory limitations when **DITA2Go** tries to run the compiler.

If the HTML Help compiler is not on your system PATH, you must tell **DITA2Go** 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
```

To have **DITA2Go** 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 §44.6 [Assembling files for distribution](#) on page 792.

See also:

§16.2.2 [Compiling and distributing Help systems](#) on page 247

§18.3.9 [Locating graphics files for HTML Help](#) on page 318

### 18.13.2 Copying output files and compiling later

You must 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 **DITA2Go Export** dialog; that option works only on files in the directory specified by [Automation]WrapPath (see §44.3 [Understanding path values for deliverables](#) on page 788). Compile from HTML Help Workshop instead.

You can use Windows copy commands to move just the compilable HTML Help files into place after converting a DITA document.

If you copy graphics files to the compilation directory, set the following option in project configuration file d2htmlhelp.ini, to remove path information from references to the graphics files:

```
[Graphics]
StripGraphPath = Yes
```

When **DITA2Go** finishes converting your document, select the .hhp file in HTML Help Workshop, and compile the project.

### 18.13.3 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 SBAppLocale from SteelBytes:

<http://www.steelbytes.com/?mid=45>

SBAppLocale 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:

```
SBAppLocale 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 SBApplocale 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.

### 18.13.4 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](http://www.ec-software.com/products_hhreg.html)

CHM files viewed from local drives are not blocked by these security features.

## 18.14 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:*

- §18.14.1 [Interlinking multiple CHM files](#) on page 348
- §18.14.2 [Synchronizing TOC references to slave CHM files](#) on page 350
- §18.14.3 [Putting up with a binary index for merged CHM files](#) on page 350
- §18.14.4 [Merging CHM files](#) on page 350
- §18.14.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 351

*See also:*

- §16.11 [Setting up a dynamic modular Help system](#) on page 280
- §18.6.3 [Linking to external files from compiled HTML Help](#) on page 325
- §18.14.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 351
- §28.6 [Linking to other files and other DITA2Go projects](#) on page 553

### 18.14.1 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 .hpx file (but not in the current CHM file) to the name of the CHM file that contains the corresponding topic.

*In this section:*

- §18.14.1.1 [Specifying the default CHM file](#) on page 348
- §18.14.1.2 [Mapping DITA files to CHM files](#) on page 349
- §18.14.1.3 [Requiring DITA2Go to use paths for mapped DITA files](#) on page 349

*See also:*

- §18.6.3 [Linking to external files from compiled HTML Help](#) on page 325
- §28.6 [Linking to other files and other DITA2Go projects](#) on page 553

Rob Chandler's Web site: <http://www.helpware.net/htmlhelp/linktochm.htm>

#### 18.14.1.1 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 §18.3.6 [Naming project and compiled files for HTML Help](#) on page 317.

**DITA2Go** uses the default CHM file name as the destination for any DITA files that have not been mapped to other CHM file names; see §18.14.1.2 [Mapping DITA files to CHM files](#) on page 349.

To support cross references between the current CHM file and CHM files from projects in other output directories, see §28.6 [Linking to other files and other DITA2Go projects](#) on page 553. To support interproject hypertext links, see §18.6.3 [Linking to external files from compiled HTML Help](#) on page 325.

### 18.14.1.2 Mapping DITA files to CHM files

For CHM files other than the default file (see §18.14.1.1 [Specifying the default CHM file](#) on page 348), specify how DITA 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 DITA files to the default CHM file.

*No file extensions* Do not include file extensions in mappings.

*Specify paths to DITA files* It is best to include a path to each DITA file, because you could have several files with the same name in different 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 DITA files, forward slashes are preferred. **DITA2Go** uses those cross-reference paths to find the referenced files. See §18.14.1.3 [Requiring DITA2Go to use paths for mapped DITA files](#) on page 349.

*Multiple paths to a single DITA file* To handle several possible paths to the same DITA 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.

### 18.14.1.3 Requiring DITA2Go to use paths for mapped DITA files

When you map DITA files to CHM files (see §18.14.1.2 [Mapping DITA files to CHM files](#) on page 349), by default **DITA2Go** first checks for the presence of a DITA file with the path specified in [ChmFiles]; if the file is not found, **DITA2Go** checks for the file without a path.

To require **DITA2Go** 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:*

§18.14.1.1 [Specifying the default CHM file](#) on page 348

### 18.14.2 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 §18.6.2 [Specifying href link syntax for HTML Help](#) on page 324.

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 **DITA2Go** 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>
```

### 18.14.3 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 **DITA2Go** 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.

### 18.14.4 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** PI marker in your main-project DITA 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** PI 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:

§16.4.3 [Setting contents levels for HTML-based Help](#) on page 251

§16.5.7 [Specifying index link destinations for HTML-based Help](#) on page 255

§18.14.5 [Comparing HHW settings for stand-alone vs. merged CHMs](#) on page 351

Rob Chandler's Web site: [http://helpware.net/htmlhelp/how\\_to\\_merge.htm](http://helpware.net/htmlhelp/how_to_merge.htm)

## 18.14.5 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 18-3 shows the rationale for certain combinations of settings for three CHM roles.

**Table 18-3 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 §18.4.2 <a href="#">Adding tabs and toolbar buttons to HTML Help</a> on page 319)</p> <p><b>Options Files</b> tab: remove .hhk file (to synchronize the TOC with the index when Binary TOC is enabled)</p> <p><b>Window Types Buttons</b> tab: check <b>Prev</b> and <b>Next</b> (for native HTML Help browse navigation)</p>

**Table 18-3 Rationale for HHW settings by CHM role (continued)**

CHM role	HTML Help Workshop settings and their effects
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> <b>Files</b> tab: retain .hhk file (or the indexes will not merge)</p> <p><i>Window Types</i> <b>Buttons</b> tab: clear <b>Prev</b> and <b>Next</b> 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> <b>Files</b> tab: retain .hhk file (or the index will not merge)</p> <p><i>Window Types</i> <b>Buttons</b> tab: clear <b>Prev</b> and <b>Next</b> check boxes (native HTML Help browse navigation does not function correctly in merged files)</p>

Table 18-4 summarizes the settings that should work for stand-alone HTML Help projects and for merged projects.

**Table 18-4 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> : <b>Files</b> tab, <b>Index</b>	Remove .hhk	Retain .hhk
<i>Window Types</i> : <b>Buttons</b> tab	Check <b>Prev</b> and <b>Next</b>	Clear <b>Prev</b> and <b>Next</b>

See also:

§18.14.4 [Merging CHM files](#) on page 350

[http://www.helpware.net/htmlhelp/how\\_to\\_merge.htm](http://www.helpware.net/htmlhelp/how_to_merge.htm)

[http://www.helpware.net/FAR/help/dlg\\_hhpedit\\_sec.htm](http://www.helpware.net/FAR/help/dlg_hhpedit_sec.htm)

[http://www.help-info.de/de/FAR/dlg\\_hhpedit.htm](http://www.help-info.de/de/FAR/dlg_hhpedit.htm)

# 19 Generating OmniHelp

---

**DITA2Go** 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 22 and sections 27 through 43 apply also. Topics include:

- §19.1 [Understanding how OmniHelp works](#) on page 353
- §19.2 [Setting up OmniHelp viewer control files](#) on page 354
- §19.3 [Setting up an OmniHelp project](#) on page 357
- §19.4 [Using CSS with OmniHelp](#) on page 361
- §19.5 [Customizing OmniHelp display features](#) on page 363
- §19.6 [Choosing navigation features for OmniHelp](#) on page 367
- §19.7 [Configuring contents and index for OmniHelp](#) on page 367
- §19.8 [Providing related-topic links in OmniHelp](#) on page 370
- §19.9 [Jumping to secondary windows in OmniHelp](#) on page 370
- §19.10 [Configuring full-text search for OmniHelp](#) on page 371
- §19.11 [Setting up CSH for OmniHelp](#) on page 375
- §19.12 [Merging OmniHelp projects](#) on page 377
- §19.13 [Assembling OmniHelp files for viewing](#) on page 380
- §19.14 [Deploying OmniHelp](#) on page 381

*See also:*

- §16 [Producing on-line Help](#) on page 243

To determine which configuration settings will produce the appearance and functionality you want, see also:

- §22 [Converting to HTML/XHTML](#) on page 429
- §27 [Splitting and extracting files](#) on page 523
- §30 [Mapping text formats to HTML/XML](#) on page 565
- §33 [Converting tables to HTML](#) on page 625

## 19.1 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 DITA 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.



<i>Help develop OmniHelp</i>	<p>Software developers are invited to contribute to the further development of OmniHelp. The OmniHelp project is officially hosted on SourceForge:</p> <p><a href="https://sourceforge.net/projects/omnihelp/">https://sourceforge.net/projects/omnihelp/</a></p> <p>The OmniHelp Design Report describes how OmniHelp was designed and built. You can read it here:</p> <p><a href="http://www.dita2go.com">http://www.dita2go.com</a></p>
<i>Modify OmniHelp</i>	<p>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 <b>DITA2Go</b> configuration parameters. <i><b>However, to undertake major modifications, you must be conversant in JavaScript, CSS, and HTML 4.</b></i></p>
<i>Maintain your modifications</i>	<p>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 <code>ohct*.css</code>, <code>oh*.js</code>, or <code>oh*.htm</code> 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 § <a href="#">New information</a> on page 17. Take advantage of any new settings to minimize JavaScript changes.</p>
<i>License OmniHelp</i>	<p>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:</p> <p><a href="http://www.gnu.org/copyleft/lesser.html">http://www.gnu.org/copyleft/lesser.html</a></p>

## 19.2 Setting up OmniHelp viewer control files

To view **DITA2Go** OmniHelp output, you will need a set of JavaScript and HTML or XHTML control files. Most of these control files are included in your **DITA2Go** distribution; the rest are generated each time you run **DITA2Go**.

*In this section:*

- § 19.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 354
- § 19.2.2 [Making OmniHelp viewer control files available](#) on page 355
- § 19.2.3 [Customizing OmniHelp viewer control files](#) on page 355
- § 19.2.4 [Examining generated control and data files](#) on page 356

### 19.2.1 Choosing XHTML vs. HTML OmniHelp control files

Your **DITA2Go** 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 19-1](#) on page 356.

**Note:** The value of `OHProjFileXhtml` determines the default value of `OHViewPath`; see §19.13 [Assembling OmniHelp files for viewing](#) on page 380.

## 19.2.2 Making OmniHelp viewer control files available

Your **DITA2Go** 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 §19.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 354), 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 **DITA2Go**. *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 §19.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 354):

<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 **DITA2Go**, 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 §19.13 [Assembling OmniHelp files for viewing](#) on page 380.

## 19.2.3 Customizing OmniHelp viewer control files

[Table 19-1](#) 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 19-1](#), 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 19-1](#). 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\ohvd2g`

Copy all files from this directory to the following location:

%OMSYSHOME%\common\local\omnihelp\ohvd2g

Modify only the files in the local directory; those in the system directory will be overwritten every time you update **DITA2Go**.

**Table 19-1 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	<a href="#">19.4</a>
	ohctn4.css	CSS for NN4 for navigation panes	Req for NN4	Yes	<a href="#">19.4</a>
	ohctn6.css	CSS for Mozilla for nav. panes	Firefox, etc.	Yes	<a href="#">19.4</a>
	ohctrl.css	Generic CSS for navigation panes	Required	Yes	<a href="#">19.4</a>
HTML (?=h) or XHTML (=?x)	o?ctrl.htm	Loader for JavaScript	Required	No	<a href="#">19.3</a>
	o?frame.htm	Frameset	Required	No	<a href="#">19.3</a>
	o?main.htm	Loading... message	Required	No	
	o?merged.htm	Run-time project merging	Optional	No	<a href="#">19.12</a>
	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	<a href="#">19.5.1</a>
JavaScript	ohctrl.js	Start-up and interfacing script	Required	No	<a href="#">19.3</a>
	ohframe.js	Frameset script	Required	No	<a href="#">19.3</a>
	ohfts.js	Search presentation script	Optional	No	<a href="#">19.6</a>
	ohidx.js	Index presentation script	Optional	No	<a href="#">19.7</a>
	o?lang.js	Text of error messages	Required	Yes	<a href="#">19.5.5</a>
	ohlangct.js	Text of control labels, etc.	Required	Yes	<a href="#">19.5.5</a>
	ohlangtp.js	Text of button labels	Required	Yes	<a href="#">19.5.5</a>
	ohmain.js	CSS-setting script for topic pane	Required	No	<a href="#">19.11</a>
	ohmerge.js	Script used in ohctrl.htm	Optional	No	<a href="#">19.12</a>
	ohmerged.js	Run-time merging script	Optional	No	<a href="#">19.12</a>
	ohrel.js	Related-topics presentation script	Optional	No	<a href="#">19.8</a>
	ohstart.js	Start-up script for project	Required	No	<a href="#">19.3</a>
	ohtoc.js	Contents presentation script	Optional	No	<a href="#">19.7</a>
	ohtop.js	Top-navigation-pane script	Required	No	<a href="#">19.5.1</a>
Image	ohlogo.jpg	OmniHelp logo	Optional	No	
	ohlc*.gif	Icons for expandable TOC view	Optional	No	<a href="#">19.7</a>
	ohvalid?.gif	W3C validation icon	Optional	No	

## 19.2.4 Examining generated control and data files

When you run **DITA2Go** with OmniHelp as the output type, **DITA2Go** produces additional data and control files, depending on your project settings. These files are listed

in [Table 19-2](#). All are placed in the project directory you specified for your OmniHelp project. *Do not rename or edit any of these files.*

**Table 19-2 OmniHelp data and control files generated by DITA2Go**

File type	File name	Content	Ref.
Data	<i>myproj_oha.js</i>	Context-sensitive-help entries	<a href="#">19.11</a>
	<i>myproj_ohc.js</i>	Contents entries	<a href="#">19.6</a>
	<i>myproj_ohk.js</i>	Index entries	<a href="#">19.6</a>
	<i>myproj_ohl.js</i>	Related-topics entries	<a href="#">19.6</a>
	<i>myproj_ohs.js</i>	Full-text-search entries	<a href="#">19.10</a>
HTML or XHTML	<i>_myproj.htm</i>	Start-up project file	<a href="#">19.3</a>
JavaScript	<i>myproj_ohx.js</i>	Project settings from <i>d2javahelp.ini</i>	<a href="#">19.3</a>

In addition to the files listed in [Table 19-2](#), when you first set up an OmniHelp project **DITA2Go** optionally generates a CSS file (default name *ohmain.css*) for topic content; see §19.4.1 [Specifying CSS for topics in OmniHelp](#) on page 361.

## 19.3 Setting up an OmniHelp project

To add or change any of the options described in this section, edit configuration file *\_d2omnihelp.ini*, located in the project directory. Edit configuration file *\_d2omnihelp.ini* to add or change any of the options described in this section.

*In this section:*

- §19.3.1 [Creating an OmniHelp project](#) on page 357
- §19.3.2 [Deciding where to locate configuration settings](#) on page 357
- §19.3.4 [Giving your OmniHelp project a title](#) on page 359
- §19.3.5 [Specifying the starting topic](#) on page 359
- §19.3.6 [Specifying memory requirements](#) on page 359
- §19.3.7 [Removing paths from interfile links for OmniHelp](#) on page 359
- §19.3.8 [Getting OmniHelp supporting files in the right place](#) on page 360

### 19.3.1 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 DITA document is located. Optionally, create a subdirectory for graphics files.
2. Copy configuration file *d2omnihelp.ini* from your **DITA2Go** *config\local* directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:
3. Use a text editor to edit the *\_d2omnihelp.ini* configuration file (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

### 19.3.2 Deciding where to locate configuration settings

When you set up an HTML Help project, if configuration file *\_d2omnihelp.ini* is not already present in the project directory, you must copy this file from your **DITA2Go** *config\local* directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29).

Which configuration file? To configure HTML Help 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	_d2omnihelp.ini	Current project directory
All OmniHelp projects	local_d2omnihelp_config.ini	%omsyshome%\d2g\local\config\projects

See §39.4 [Deciding which configuration file to edit](#) on page 734.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §22 [Converting to HTML/XHTML](#) on page 429
- §27 [Splitting and extracting files](#) on page 523
- §30 [Mapping text formats to HTML/XML](#) on page 565
- §32 [Including graphics in HTML](#) on page 611
- §33 [Converting tables to HTML](#) on page 625

### 19.3.3 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 DITA document. **DITA2Go** uses the value of `ProjectName` for the following purposes:

- generated-data-file base names: *myproj.oh\**
- project identifier, when OmniHelp projects are merged; see §19.12 [Merging OmniHelp projects](#) on page 377.
- 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:

§19.3.4 [Giving your OmniHelp project a title](#) on page 359

### 19.3.4 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”.

### 19.3.5 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.

### 19.3.6 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.

### 19.3.7 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:

§28.6.1 [Retaining file paths in interfile links](#) on page 553

## 19.3.8 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 **DITA2Go** generates from your document. Supporting files include:

[Viewer and control files](#)

[Graphics files](#)

[Optional files.](#)

*Viewer and control files*

After you run **DITA2Go**, control files and viewer files must be copied from the viewer directory (see §19.2 [Setting up OmniHelp viewer control files](#) on page 354) to the final distribution directory for your project. **DITA2Go** can do this for you; see §19.13 [Assembling OmniHelp files for viewing](#) on page 380.

To view OmniHelp, the view directory must contain the following:

- all files marked Required in column **View?** in [Table 19-1](#) on page 356 (including either `oh*.htm` or `ox*.htm`, depending on the start-up file type; see §19.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 354).
- all files listed in [Table 19-2](#) on page 357.

*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 **DITA2Go** to fetch your referenced graphics:

```
[Automation]
WrapAndShip = Yes
CopyOriginalGraphics = Yes
```

When `CopyOriginalGraphics=Yes`, **DITA2Go** follows the file paths in your DITA source to find the graphics files to copy.

To tell **DITA2Go** 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 §44.3 [Understanding path values for deliverables](#) on page 788). 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, **DITA2Go** converts them to forward slashes before inserting the references in your HTML output. If you specify `CopyOriginalGraphics=Yes`, **DITA2Go** copies graphics files to the directory specified by `GraphPath`, after generating HTML files.

See also:

§19.13 [Assembling OmniHelp files for viewing](#) on page 380.

§32.1 [Locating graphics files for HTML](#) on page 611

§44.7 [Placing graphics files for distribution](#) on page 796

*Optional files*

A browser loads optional files (marked Optional under **View?** in [Table 19-1](#) on page 356), 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 §19.2 [Setting up OmniHelp viewer control files](#) on page 354.

## 19.4 Using CSS with OmniHelp

OmniHelp relies on CSS (cascading style sheets), because **DITA2Go** 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:*

§19.4.1 [Specifying CSS for topics in OmniHelp](#) on page 361

§19.4.2 [Understanding how CSS works in OmniHelp topics](#) on page 362

§19.4.3 [Specifying CSS for OmniHelp navigation frames](#) on page 362

### 19.4.1 Specifying CSS for topics in OmniHelp

When you set up a new OmniHelp project (see §19.3.1 [Creating an OmniHelp project](#) on page 357), you can name a default CSS file for the topic frame; the default name of this default file is `ohmain.css`. **DITA2Go** generates `ohmain.css` (or whatever name you specify) and places it in the project directory the first time you convert your document; see §31.3 [Understanding how DITA2Go generates CSS](#) on page 592.

At set-up time **DITA2Go** includes the following CSS-related entries in newly created configuration file `d2omnihelp.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 §31.4 [Specifying CSS file and link options](#) on page 593).

*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 §31.6 [Linking to alternate CSS files](#) on page 597.

*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 §44.6 [Assembling files for distribution](#) on page 792), even though it will not be used. And if you remove that file from the project directory, **DITA2Go** 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 §31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593.

## 19.4.2 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 19-2](#) on page 357), which are based on `[OmniHelpOptions]` settings in the configuration file (see §19.4.1 [Specifying CSS for topics in OmniHelp](#) on page 361). 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 **DITA2Go** 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.

## 19.4.3 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>

## 19.5 Customizing OmniHelp display features

*In this section:*

§19.5.1 [Configuring OmniHelp window usage and frameset dimensions](#) on page 363

§19.5.2 [Altering OmniHelp top navigation frame content](#) on page 364

§19.5.3 [Modifying OmniHelp navigation aids](#) on page 364

§19.5.4 [Choosing whether to use cookies for OmniHelp](#) on page 365

§19.5.5 [Localizing the OmniHelp interface](#) on page 365

§19.5.6 [Modifying OmniHelp CSS classes](#) on page 365

§19.5.7 [Modifying the OmniHelp template](#) on page 366

### 19.5.1 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 §19.14.3 [Coping with browser quirks](#) on page 381.

*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 §19.9 [Jumping to secondary windows in OmniHelp](#) on page 370. For more information, look up the `window.open()` function in any JavaScript reference.

## 19.5.2 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 §19.5.1 [Configuring OmniHelp window usage and frameset dimensions](#) on page 363), 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 §19.1 [Understanding how OmniHelp works](#) on page 353.

## 19.5.3 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.

### 19.5.4 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.

### 19.5.5 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

### 19.5.6 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 **DITA2Go** applies to the top navigation table. JavaScript in `ohtop.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;&nbsp;&nbsp;OmniHelp &nbsp;&nbsp;&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:

ohctie.css	Internet Explorer
ohctn4.css	Netscape 4.x
ohctn6.css	Mozilla-derived browsers, such as Firefox
ohctrl.css	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.

## 19.5.7 Modifying the OmniHelp template

Your DITA2Go distribution directory contains a copy of file ohtml.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 §19.1 [Understanding how OmniHelp works](#) on page 353.

You can copy ohtml.ini to your project directory. If you are brave, you can specify a path to ohtml.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=ohtml.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\ohvd2g
```

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`.

## 19.6 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 19-3 lists the navigation features; all are displayed in the left-hand frame:

**Table 19-3** OmniHelp navigation features

Feature	NavElems value	Reference
Contents	Toc	§16.4 <a href="#">Configuring contents entries for Help systems</a> on page 250 §19.7 <a href="#">Configuring contents and index for OmniHelp</a> on page 367
Index	Idx	§16.5 <a href="#">Configuring index entries for Help systems</a> on page 251 §19.7 <a href="#">Configuring contents and index for OmniHelp</a> on page 367
Full-text search	Fts	§19.10 <a href="#">Configuring full-text search for OmniHelp</a> on page 371
Related topics	Rel	§19.8 <a href="#">Providing related-topic links in OmniHelp</a> on page 370

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 **DITA2Go** 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), or None.
RefFileType=Full
```

`RefFileType` values have the following effects:

**Full** *Default for single files.* **DITA2Go** creates a set of `myproj.oh*` files for the original DITA file.

**None** No `myproj.oh*` files nor `DCL .bh*` files are produced.

*See also:*

§16.3.1 [Modifying contents or index production for HTML-based Help](#) on page 249.

§19.7 [Configuring contents and index for OmniHelp](#) on page 367

## 19.7 Configuring contents and index for OmniHelp

*In this section:*

§19.7.1 [Understanding OmniHelp contents and index creation](#) on page 368

§19.7.2 [Choosing whether to use expanding contents or index](#) on page 368

§19.7.3 [Choosing how far to expand contents and index subentries](#) on page 368

§19.7.4 [Providing alternate expansion icons for contents or index](#) on page 369

§19.7.5 [Excluding Open All and Close All buttons](#) on page 369

## 19.7.1 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:

§16.4.2 [Including contents entries in HTML-based Help](#) on page 250.

§16.4.3 [Setting contents levels for HTML-based Help](#) on page 251.

When you click links in the topic pane while the contents pane is displayed, the contents pane stays synchronized with whatever topic you visit.

**DITA2Go** creates an OmniHelp index from the index entries in your DITA 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 §16.5 [Configuring index entries for Help systems](#) on page 251.

## 19.7.2 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
```

## 19.7.3 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.

## 19.7.4 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
```

## 19.7.5 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
```

## 19.7.6 Redirecting *See* and *See also* index entries

**DITA2Go** redirects *See* and *See also* references for the OmniHelp index. On generating OmniHelp output, **DITA2Go** points each such link to the referenced entry *in the OmniHelp index* (rather than in the topic where the index term appeared in DITA).

*See also:*

§16.5.6.1 [Identifying \*See\* and \*See also\* index references](#) on page 254

§16.5.6.3 [Choosing where to sort \*See also\* index references](#) on page 255

## 19.8 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 §16.6 [Providing related-topic links for Help systems](#) on page 258 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 §19.6 [Choosing navigation features for OmniHelp](#) on page 367), 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:*

§16.6.2 [Understanding how ALinks work](#) on page 259

§16.6.4 [Adding related-topic link keywords in DITA XML](#) on page 260

§16.6.5 [Adding ALink and KLink jumps in DITA XML](#) on page 261

§16.6.6 [Creating target-and-jump ALinks for HTML-based Help](#) on page 262

## 19.9 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:*

§16.7 [Jumping to secondary windows in Help systems](#) on page 262

§16.8 [Creating pop-up topics for Help systems](#) on page 263

## 19.10 Configuring full-text search for OmniHelp

After generating OmniHelp output, by default **DITA2Go** 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:*

§19.10.1 [Understanding how OmniHelp FTS works](#) on page 372

§19.10.2 [Generating search data](#) on page 372

§19.10.3 [Making compound terms searchable](#) on page 372

§19.10.4 [Supporting search for non-ANSI text](#) on page 373

§19.10.5 [Specifying length of search terms](#) on page 373

§19.10.6 [Excluding search terms](#) on page 373

§19.10.7 [Excluding content from being searched](#) on page 374

§19.10.8 [Using regular expressions in search](#) on page 374

§19.10.9 [Highlighting search terms found in topics](#) on page 374

## 19.10.1 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 §19.10.6 [Excluding search terms](#) on page 373), 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 §19.1 [Understanding how OmniHelp works](#) on page 353.

## 19.10.2 Generating search data

DITA2Go 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 §19.6 [Choosing navigation features for OmniHelp](#) on page 367) 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.

## 19.10.3 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.

#### 19.10.4 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`, **DITA2Go** will use these DLLs to prepare your OmniHelp output, depending on the value you specify for `UnicodeLocale`.

#### 19.10.5 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
```

#### 19.10.6 Excluding search terms

When you generate OmniHelp, **DITA2Go** builds a search index that includes all the words in the converted files, except for a list of words to be excluded. **DITA2Go** 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`, **DITA2Go** excludes from the search index the default words and any words listed in the `[StopWords]` section of your configuration file.

When `UseDefaultStopWords=No`, **DITA2Go** 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.

### 19.10.7 Excluding content from being searched

DITA topicref attribute `@search` (with value `yes` or `no`) allows you to specify whether the referenced topic is to be included in full-text search. **DITA2Go** excludes from OmniHelp full-text search topics that are referenced with `@search="no"`.

*Do it with PI markers*

If your DITA maps do not include values for the topicref `@search` attribute, you can achieve the same effect with PI markers inserted in the topicrefs, and a ditaval file. For example:

```
<?dthtm Search="no" ?>
```

In this example, the effect on **DITA2Go** output is the same as if you had included `@search="no"` in the topicref for each such topic. You can get the same effect by placing the PI marker inside the `<title>` element of the topic.

*Paragraph-level granularity*

You can achieve paragraph-level search granularity with PI markers placed within a topic; the last **Search** value in a paragraph applies to the entire paragraph.

### 19.10.8 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.

### 19.10.9 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.

## 19.11 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 `TopicAlias` PI markers in your DITA 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:*

§19.11.1 [Specifying alias prefixes for OmniHelp CSH calls](#) on page 375

§19.11.2 [Referencing OmniHelp topic IDs from an application](#) on page 375

§19.11.3 [Using redirect pages for OmniHelp CSH calls](#) on page 376

§19.11.4 [Executing browser commands for OmniHelp CSH calls](#) on page 376

*See also:*

§16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277

### 19.11.1 Specifying alias prefixes for OmniHelp CSH calls

**DITA2Go** produces CSH alias entries from every `TopicAlias` PI marker in your DITA document whose content starts with one of the prefixes you specify. If you do not specify any prefixes, all `TopicAlias` PI 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 `TopicAlias` PI marker in your document whose text starts with `HIDC_` or `IDH_`.

**DITA2Go** always creates the alias file, needed even if empty.

### 19.11.2 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 §19.3 [Setting up an OmniHelp project](#) on page 357.

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.

### 19.11.3 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 `TopicAlias` PI 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/_dita2go.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 §19.11.2 [Referencing OmniHelp topic IDs from an application](#) on page 375.

### 19.11.4 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 §19.11.3 [Using redirect pages for OmniHelp CSH calls](#) on page 376).

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
```

- 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 `TopicAlias` PI marker in DITA XML. Notice the *three* forward slashes in the file reference. This syntax should open the correct OmniHelp topic file.

## 19.12 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:*

§19.12.1 [Understanding the OmniHelp merge process](#) on page 377

§19.12.2 [Listing and mapping OmniHelp subprojects](#) on page 378

§19.12.3 [Providing TOC placeholders for OmniHelp subprojects](#) on page 378

§19.12.4 [Deciding when to merge OmniHelp subprojects](#) on page 379

*See also:*

§16.11 [Setting up a dynamic modular Help system](#) on page 280

### 19.12.1 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.

*Project names must be unique*

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.

*Subprojects can be nested*

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.

*Main project must know about subprojects*

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 §19.12.2 [Listing and mapping OmniHelp subprojects](#) on page 378), and each subproject must have an entry in the including project's table of contents (see §19.12.3 [Providing TOC placeholders for OmniHelp subprojects](#) on page 378).

## 19.12.2 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.

## 19.12.3 Providing TOC placeholders for OmniHelp subprojects

Place a **HelpMerge** PI marker in your main-project DITA 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** PI 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** PI marker at the very beginning of the main project, and do not include duplicate **HelpMerge** PI 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 **DITA2Go**, 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:

<i>n</i>	Contents level for the subproject entry in the master-project contents
<i>title</i>	Title of the subproject
<i>name</i>	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:

§16.4.3 [Setting contents levels for HTML-based Help](#) on page 251

§19.12.2 [Listing and mapping OmniHelp subprojects](#) on page 378

## 19.12.4 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 §19.12.2 [Listing and mapping OmniHelp subprojects](#) on page 378.
- The main project's table of contents includes a merge point for the subproject; see §19.12.3 [Providing TOC placeholders for OmniHelp subprojects](#) on page 378.
- 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 §19.11 [Setting up CSH for OmniHelp](#) on page 375). 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.

## 19.13 Assembling OmniHelp files for viewing

By default, **DITA2Go** copies all files with the following extensions from the project directory to the wrap directory (see §44.2 [Activating and logging production of deliverables](#) on page 788):

```
*.htm *.css *.js *.gif *.jpg *.png
```

You can change the list of files to be copied; see §44.6 [Assembling files for distribution](#) on page 792. **DITA2Go** 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 §19.2.2 [Making OmniHelp viewer control files available](#) on page 355. 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. **DITA2Go** copies the files listed in [Table 19-4](#) from the directory designated by OHViewPath to the directory designated by WrapPath, if specified, otherwise to the project directory.

**Table 19-4** 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 §19.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 354.

To have **DITA2Go** 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 19-4](#). 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 §19.2.1 [Choosing XHTML vs. HTML OmniHelp control files](#) on page 354.

See also:

- §16.2.2 [Compiling and distributing Help systems](#) on page 247
- §44.6 [Assembling files for distribution](#) on page 792

## 19.14 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:*

§19.14.1 [Starting with the default topic or a specified topic](#) on page 381

§19.14.2 [Restarting where you left off](#) on page 381

§19.14.3 [Coping with browser quirks](#) on page 381

### 19.14.1 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 §19.11 <a href="#">Setting up CSH for OmniHelp</a> on page 375.

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 §19.3 [Setting up an OmniHelp project](#) on page 357.

### 19.14.2 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**.

### 19.14.3 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 **DITA2Go** 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 §19.5.1 [Configuring OmniHelp window usage and frameset dimensions](#) on page 363), 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 §19.14.1 [Starting with the default topic or a specified topic](#) on page 381).

*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.



# 20 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 22 and sections 27 through 43 apply also. Topics include:

- §20.1 [Deciding which Java Help system to use](#) on page 385
- §20.2 [Obtaining tools for a Java-based Help system](#) on page 385
- §20.3 [Setting up a JavaHelp or Oracle Help project](#) on page 386
- §20.4 [Generating contents and index](#) on page 395
- §20.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 397
- §20.6 [Creating and viewing a Java Archive \(JAR\) file](#) on page 400
- §20.7 [Converting a glossary to JavaHelp 2](#) on page 401
- §20.8 [Defining windows for JavaHelp or Oracle Help](#) on page 403
- §20.9 [Linking to destinations within topics](#) on page 409
- §20.10 [Creating ALinks for Oracle Help](#) on page 409
- §20.11 [Merging JavaHelp or Oracle Help systems](#) on page 410
- §20.12 [Setting up CSH for JavaHelp or Oracle Help](#) on page 410

*See also:*

- §16 [Producing on-line Help](#) on page 243

## 20.1 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.

**DITA2Go** 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 **DITA2Go** 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.

## 20.2 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.

## 20.3 Setting up a JavaHelp or Oracle Help project

*In this section:*

- §20.3.1 [Creating a JavaHelp or Oracle Help for Java project](#) on page 386
- §20.3.2 [Deciding where to locate configuration settings](#) on page 386
- §20.3.3 [Specifying output options for JavaHelp](#) on page 387
- §20.3.4 [Establishing a JavaHelp environment](#) on page 387
- §20.3.5 [Establishing an Oracle Help environment](#) on page 388
- §20.3.6 [Creating a directory structure for JavaHelp / Oracle Help](#) on page 389
- §20.3.7 [Configuring the helpset file](#) on page 392
- §20.3.8 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 394
- §20.3.9 [Compiling JavaHelp with Helen](#) on page 395

### 20.3.1 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 DITA document is located.
2. Copy the appropriate configuration file, `_d2javahelp.ini` or `_d2oraclehelp.ini`, from your **DITA2Go** `config\local` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:
3. Use a text editor to edit the `_d2javahelp.ini` or `_d2oraclehelp.ini` configuration file (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

### 20.3.2 Deciding where to locate configuration settings

When you set up a JavaHelp or Oracle Help project, if configuration file `_d2javahelp.ini` (or `_d2oraclehelp.ini`) is not already present in your project directory, you must copy this file from your **DITA2Go** `config\local` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29).

*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	_d2javahelp.ini	Current project directory
All JavaHelp projects	local_d2javahelp_config.ini	%omsyshome%\d2g\local\config\projects

For *Oracle Help* output:

<u>Scope</u>	<u>Configuration file</u>	<u>Location</u>
Current project only	_d2oraclehelp.ini	Current project directory
All Oracle Help projects	local_d2oraclehelp_config.ini	%omsyshome%\d2g\local\config\projects

See §39.4 [Deciding which configuration file to edit](#) on page 734.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §22 [Converting to HTML/XHTML](#) on page 429
- §27 [Splitting and extracting files](#) on page 523
- §30 [Mapping text formats to HTML/XML](#) on page 565
- §32 [Including graphics in HTML](#) on page 611
- §33 [Converting tables to HTML](#) on page 625

### 20.3.3 Specifying output options for JavaHelp

By default, **DITA2Go** 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:

- §20.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 397
- §20.6 [Creating and viewing a Java Archive \(JAR\) file](#) on page 400

### 20.3.4 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 §20.2 [Obtaining tools for a Java-based Help system](#) on page 385.

*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 **DITA2Go** 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.

## 20.3.5 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 §20.2 [Obtaining tools for a Java-based Help system](#) on page 385.

*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 §20.3.4 [Establishing a JavaHelp environment](#) on page 387), 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"
oracle.help.demo.ChoiceDemo "G:\OmniSys\UG\ohj\help\ugdita2go.hs"
%*
```

Your experience might be different.

## 20.3.6 Creating a directory structure for JavaHelp / Oracle Help

*In this section:*

- §20.3.6.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 389
- §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389
- §20.3.6.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 391
- §20.3.6.4 [Specifying a path for search-index links](#) on page 391
- §20.3.6.5 [Manually copying and deleting output files](#) on page 392

### 20.3.6.1 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 **DITA2Go** project directory. You can create the directory structure, or let **DITA2Go** do it for you; see §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389. 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>	DITA document files
<i>..MyOutput</i>	<b>DITA2Go</b> output files (normal <b>DITA2Go</b> project directory)
<i>Top JH or OHJ level starts here:</i>	
...help	Help files copied from <i>MyOutput</i> : <code>.hs</code> , <code>.jhm</code> , <code>.xml</code>
.....graphics	Image files copied from <i>MyOutput</i> (or another location)
.....html	<code>.htm</code> and <code>.css</code> files copied from <i>MyOutput</i>

After **DITA2Go** generates output, the helpset file (`.hs`) and navigational files (`.jhm` and `.xml`) are copied from the **DITA2Go** project directory to the `help` directory.

### 20.3.6.2 Letting DITA2Go set up the directory structure and copy files

To have **DITA2Go** 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, **DITA2Go** 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 §20.3.6.4 [Specifying a path for search-index links](#) on page 391.

The directory designated by GraphSubdir is the default JavaHelp and Oracle Help setting for [Graphics]GraphPath, with “./” prepended; see §20.3.6.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 391.

*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, **DITA2Go** automatically populates the directory structure. After generating HTML files and optionally creating a full-text search index, **DITA2Go** 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 **DITA2Go** copy graphics files from a location other than the project directory:

```
[Automation]
WrapAndShip = Yes
CopyOriginalGraphics = Yes
```

When `CopyOriginalGraphics=Yes`, **DITA2Go** follows the file paths in your DITA source to find the graphics files to copy to the directory specified by `GraphSubdir`.

See also:

§16.2.2 [Compiling and distributing Help systems](#) on page 247.

§44 [Producing deliverable results](#) on page 787

### 20.3.6.3 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 §20.3.6.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 389).

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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389
- 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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389) 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 §3.4 [Understanding the rules for configuration settings](#) on page 62.

**DITA2Go** uses the value of `GraphPath` for the `src` attribute of `<img>` tags. See §40.2.1.1 [Specifying graphics location for HTML](#) on page 747.

To specify the location of images assigned to specific JavaHelp 2 windows, see §20.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 403.

### 20.3.6.4 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 §20.3.6.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 389.

**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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389) 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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

### 20.3.6.5 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 **DITA2Go** 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.

## 20.3.7 Configuring the helpset file

*In this section:*

- §20.3.7.1 [Specifying helpset file name and title](#) on page 392
- §20.3.7.2 [Specifying a default starting topic for the helpset](#) on page 393
- §20.3.7.3 [Deciding whether to rewrite the helpset file](#) on page 393
- §20.3.7.4 [Providing a “favorites” option for JavaHelp 2](#) on page 393
- §20.3.7.5 [Adding custom helpset sections for JavaHelp 2](#) on page 393
- §20.3.7.6 [Requiring full paths in the helpset file](#) on page 394

### 20.3.7.1 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 DITA document file.

To have **DITA2Go** 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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

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.

### 20.3.7.2 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 internally generated object ID 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/dita2go.htm" />
```

If you do not specify a value for DefaultTopic, **DITA2Go** tries to autodetect the target name; however, this works only if **DITA2Go** is rewriting the helpset file *after* the first time you ran the conversion. **DITA2Go** uses the first topic ID in the first file in the book. If all else fails, **DITA2Go** uses the base name of the .bookmap or .ditamap file.

### 20.3.7.3 Deciding whether to rewrite the helpset file

**DITA2Go** creates a helpset file for you the first time you run a JavaHelp conversion. Although **DITA2Go** rewrites .jhm and .xml files every time you run the conversion, by default **DITA2Go** does not rewrite the helpset file during subsequent conversion runs.

To have **DITA2Go** 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 **DITA2Go**; see §20.3.7.5 [Adding custom helpset sections for JavaHelp 2](#) on page 393.

### 20.3.7.4 Providing a “favorites” option for JavaHelp 2

If you are using **DITA2Go** 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
```

### 20.3.7.5 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 **DITA2Go** does not rewrite the helpset file each time (see §20.3.7.3 [Deciding whether to rewrite the helpset file](#) on page 393), you could simply add custom sections directly to the helpset file.

### 20.3.7.6 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 §20.3.6.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 389). The only reason to put navigational files elsewhere would be to accommodate a proprietary standard; *this setting is intended to support that use only*.

## 20.3.8 Coping with JavaHelp / Oracle Help viewer limitations

JavaHelp viewer limitations and defects are described in the *JavaHelp System User's Guide*. **DITA2Go** 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:

[Anchor tags in JavaHelp](#)

[Image size units in JavaHelp](#)

[CSS in JavaHelp or Oracle Help](#)

[Index entries](#).

*Anchor tags in  
JavaHelp*

Each anchor tag in HTML, including the <a> tag produced from each **HyperAnchor** PI marker in your DITA document, is replaced by a space in the JavaHelp viewer. There is no feasible workaround for this defect. **DITA2Go** usually produces more than one <a name= . . . >, 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 **DITA2Go** omits the suffix. Make sure you do not override this default; see §32.8.3 [Specifying px units for graphics sized in pixels](#) on page 621.



*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 §30.6.3 [Including or excluding font tags](#) on page 578.

JavaHelp CSS does not respect the list-style rule; therefore, by default, **DITA2Go** 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 §30.11.2.6 [Including or excluding the type list attribute](#) on page 588.

JavaHelp CSS does not support conditional flagging; see §9.3 [Including flags for ditaval conditions](#) on page 165.

*Index entries*

Index entries have limitations in both viewers; see §20.4.3 [Configuring index entries for JavaHelp or Oracle Help](#) on page 396.

## 20.3.9 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.

## 20.4 Generating contents and index

To understand whether, how, and when **DITA2Go** generates contents and index files for JavaHelp and Oracle Help, see §16.3.1 [Modifying contents or index production for HTML-based Help](#) on page 249.

*In this section:*

- §20.4.1 [Configuring contents entries for JavaHelp or Oracle Help](#) on page 395
- §20.4.2 [Assigning TOC images and expansion levels in JavaHelp 2](#) on page 396
- §20.4.3 [Configuring index entries for JavaHelp or Oracle Help](#) on page 396
- §20.4.4 [Locating JavaHelp or Oracle Help contents and index files](#) on page 397

*See also:*

- §16.3 [Producing contents and index for Help systems](#) on page 248
- §16.4 [Configuring contents entries for Help systems](#) on page 250
- §16.5 [Configuring index entries for Help systems](#) on page 251

### 20.4.1 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:

- §16.4.2 [Including contents entries in HTML-based Help](#) on page 250.
- §16.4.3 [Setting contents levels for HTML-based Help](#) on page 251.

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 §20.9 [Linking to destinations within topics](#) on page 409.

## 20.4.2 Assigning TOC images and expansion levels in JavaHelp 2

**DITA2Go** 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 §20.8.1.2 [Mapping image names to graphics files](#) on page 404.

## 20.4.3 Configuring index entries for JavaHelp or Oracle Help

**DITA2Go** 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 §16.5 [Configuring index entries for Help systems](#) on page 251, *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 **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** handles multiple-reference index entries for JavaHelp.

See §16.5 [Configuring index entries for Help systems](#) on page 251.

## 20.4.4 Locating JavaHelp or Oracle Help contents and index files

When **DITA2Go** 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 §20.3.6.1 [Understanding the JavaHelp / Oracle Help directory structure](#) on page 389.

## 20.5 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:*

§20.5.1 [Including a search-index link in the helpset file](#) on page 397

§20.5.2 [Creating a search index for JavaHelp](#) on page 398

§20.5.3 [Creating a search index for Oracle Help](#) on page 399

### 20.5.1 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`, **DITA2Go** 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 **DITA2Go** run the program, or you can run it yourself; see:

§20.5.2 [Creating a search index for JavaHelp](#) on page 398

§20.5.3 [Creating a search index for Oracle Help](#) on page 399

**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`, **DITA2Go** 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.

## 20.5.2 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 **DITA2Go** 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 §20.5.1 [Including a search-index link in the helpset file](#) on page 397.

Let **DITA2Go**  
create FTS

To have **DITA2Go** 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, **DITA2Go** 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`, **DITA2Go** copies all needed files from the project directory to the JavaHelp directory structure before running JHIndexer; see §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

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      (DITA files)
..MyOutput (DITA2Go files)
...Help    (JHM, HS, TOC, Index; run JHIndexer from this directory)
.....HTML
.....Graphics
```

Your structure ends up looking like this:

```
MyDoc      (DITA files)
..MyOutput (DITA2Go 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:

§16.2.2 [Compiling and distributing Help systems](#) on page 247

§20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389

§20.3.6.4 [Specifying a path for search-index links](#) on page 391

§20.5.1 [Including a search-index link in the helpset file](#) on page 397

§44.10 [Gathering and processing Help-system files](#) on page 802

## 20.5.3 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 §20.3.5 [Establishing an Oracle Help environment](#) on page 388.

You can create a search index for Oracle Help in any of the following ways:

§20.5.3.1 [Directing DITA2Go to create full-text search for Oracle Help](#) on page 399

§20.5.3.2 [Creating full-text search for Oracle Help via command line](#) on page 400

§20.5.3.3 [Using the Oracle Help Wizard to create full-text search](#) on page 400

See also:

§16.2.2 [Compiling and distributing Help systems](#) on page 247

§20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389

§20.5.1 [Including a search-index link in the helpset file](#) on page 397

§44.10 [Gathering and processing Help-system files](#) on page 802

### 20.5.3.1 Directing DITA2Go to create full-text search for Oracle Help

To have **DITA2Go** 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, **DITA2Go** 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.

If you specify a value for [Automation]WrapPath, **DITA2Go** copies all needed files from the project directory to the Oracle Help directory structure before running Indexer; see §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

### 20.5.3.2 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. Prefix options with a dash (“-”) rather than a forward slash.

### 20.5.3.3 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.

## 20.6 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:*

§20.6.1 [Creating a JAR file](#) on page 400

§20.6.2 [Viewing a JAR file](#) on page 401

### 20.6.1 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 DITA2Go create the JAR file](#)

[Create the JAR file yourself.](#)

*Let DITA2Go  
create the JAR  
file*

To have **DITA2Go** 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. **DITA2Go** provides the cvf and \* parameters, and gives the resulting JAR file the base name of your helpset file as specified by HSFileName (see §20.3.7.1 [Specifying helpset file name and title](#) on page 392), 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 *
```

## 20.6.2 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 §20.3.4 [Establishing a JavaHelp environment](#) on page 387).
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 DITA document to lowercase, and set [HTMLOptions]MakeFileHrefsLower=Yes; see §28.2.6 [Forcing link text to lowercase](#) on page 549.
- Inspect every cross-reference and hypertext PI marker in your document, and either fix the markers or change the case of the DITA 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:

§20.3.5 [Establishing an Oracle Help environment](#) on page 388.

## 20.7 Converting a glossary to JavaHelp 2

If your DITA document includes a glossary that consists of alternating terms and their definitions, you can take advantage of JavaHelp 2 glossary support.

*In this section:*

§20.7.1 [Evaluating glossary usability](#) on page 402

§20.7.2 [Assigning glossary properties](#) on page 402

§20.7.3 [Configuring glossary IDs](#) on page 402

§20.7.4 [Eliminating glossary entries from the JavaHelp TOC](#) on page 403

## 20.7.1 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.

## 20.7.2 Assigning glossary properties

To convert a DITA glossary to a JavaHelp 2 glossary:

```
[JavaHelpOptions]
; UseGlossary = No (default) or Yes (affects HelpSet File rewrite)
UseGlossary = Yes
```

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.

## 20.7.3 Configuring glossary IDs

For each glossary term, **DITA2Go** 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 =
```

**DITA2Go** stores glossary terms, IDs, and glossary file names (allowing for splitting a glossary file) in a *ChapName.bhg* file for each *.ditamap* file in your document.

**DITA2Go** 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.



## 20.7.4 Eliminating glossary entries from the JavaHelp TOC

Material that **DITA2Go** 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 DITA glossary.

To eliminate a TOC entry for the glossary, do the following:

- Assign a distinct `@outputclass` to the heading in DITA XML, and in the configuration file assign `[HTMLParaStyles]` property `Delete` to the resulting format; see §30.2.6 [Eliminating unwanted paragraphs](#) on page 569.

You must also remove any cross references to the heading from other topics.

## 20.8 Defining windows for JavaHelp or Oracle Help

*In this section:*

- §20.8.1 [Specifying window parameters for JavaHelp 2](#) on page 403
- §20.8.2 [Specifying window parameters for Oracle Help](#) on page 408
- §20.8.3 [Jumping to secondary windows in JavaHelp or Oracle Help](#) on page 408

### 20.8.1 Specifying window parameters for JavaHelp 2

**DITA2Go** 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 §20.8.2 [Specifying window parameters for Oracle Help](#) on page 408.

*In this section:*

- §20.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 403
- §20.8.1.2 [Mapping image names to graphics files](#) on page 404
- §20.8.1.3 [Understanding JavaHelp 2 window-access limitations](#) on page 404
- §20.8.1.4 [Specifying window-access object properties](#) on page 405
- §20.8.1.5 [Overriding window-access properties with markers](#) on page 407
- §20.8.1.6 [Designing your own window-access marker names](#) on page 407

#### 20.8.1.1 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 §16.8 [Creating pop-up topics for Help systems](#) on page 263. 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 **DITA2Go** 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 = DITA2Go 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 §20.8.1.4 [Specifying window-access object properties](#) on page 405.

### 20.8.1.2 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

```

### 20.8.1.3 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 assigned to an inline element in DITA XML (see §16.8.2 [Defining a pop-up hotspot](#) on page 264) looks like a hotspot in JavaHelp 2, but has no effect.

The only way to include link-specific hotspot text in DITA XML that both looks and acts like a hotspot in JavaHelp 2 is to insert in your document special PI markers that contain the hotspot text, plus (if necessary) additional special PI markers that designate font properties for hotspot text; see §20.8.1.5 [Overriding window-access properties with markers](#) on page 407.

In other words, for an active-link text hotspot, you have to use PI 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 **DITA2Go** produces is not a text object, but instead a button that immediately precedes text that is already designated as a hotspot.

#### 20.8.1.4 Specifying window-access object properties

You specify properties for window-access objects by assigning values to object-property keywords in section [JavaHelpOptions]. [Table 20-1](#) on page 405 and [Table 20-2](#) on page 406 list the keywords, the values you can assign to each keyword, and the default when you do not assign a value.

*In this section:*

§20.8.1.4.1 [Changing window type, size, or position via access object](#) on page 405

§20.8.1.4.2 [Specifying link properties via window-access object](#) on page 406

##### 20.8.1.4.1 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 §20.8.1.5 [Overriding window-access properties with markers](#) on page 407.

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 20-1](#) override the default size and position parameters in the helpset file for *all* secondary windows, making the default values moot. See §20.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 403. The only way to configure different sizes and positions for different secondary window types is by inserting special markers before each jump; see §20.8.1.5 [Overriding window-access properties with markers](#) on page 407.

*Override  
secondary  
window type*

The default value for [JavaHelpOptions]SecName is empty (see [Table 20-1](#)). 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:

§16.7 [Jumping to secondary windows in Help systems](#) on page 262

§20.8.1.1 [Assigning default window parameters for JavaHelp 2](#) on page 403

**Table 20-1 [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	

#### 20.8.1.4.2 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 20-2](#) on page 406 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 20-2](#), 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 20-2** [*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 <a href="#">Text</a> is the label When Type=Graphic, the value of keyword <a href="#">Text</a> (or keyword <a href="#">Graphic</a> ) is the location of the image file When Type=Text, Font* properties apply to the value of keyword <a href="#">Text</a>
Graphic	<i>URL for image file</i>	Default is ../graphics/lp.gif When <a href="#">Type</a> =Graphic, value is the relative URL of the GIF or JPEG file
Text	<i>plain text</i> &lt; &gt; &amp;	Default is &gt; When Type=Text, Font* properties apply to the value of keyword <a href="#">Text</a> When Type=Graphic, value can be the location of the image (which can be specified as a value either for keyword <a href="#">Text</a> or for keyword <a href="#">Graphic</a> )
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
FontSize	xx-small, x-small, small (default), medium, large, x-large  <i>index number</i> bigger smaller +n -n npt	<i>index number</i> is a plain digit bigger increases the size by one index smaller decreases the size by one index +n increases the size by <i>n</i> -n decreases the size by <i>n</i> npt 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		

<i>Eschew Type=Text</i>	Despite the limited font-tweaking possibilities listed in <a href="#">Table 20-2</a> , you might want to avoid setting <code>PopType=Text</code> or <code>SecType=Text</code> , unless you are happy with a link that consists of a single <code>&gt;</code> , <code>&lt;</code> , or <code>&amp;</code> character, or a little box (what you get when, for example, you specify <code>PopFontFamily=Symbol</code> and choose a character from the Symbol font). See §20.8.1.3 <a href="#">Understanding JavaHelp 2 window-access limitations</a> on page 404.
<i>Multiple markers for each hotspot</i>	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 §20.8.1.5 <a href="#">Overriding window-access properties with markers</a> on page 407.
<i>Not all “special” characters work</i>	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.

### 20.8.1.5 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 PI marker somewhere in your document before the link you want to tweak.

<i>Marker name</i>	The name of the PI marker is the keyword for the property to be tweaked, prefixed by <b>JH2Pop</b> for a pop-up window or <b>JH2Sec</b> for a secondary window, or by a prefix you specify; see §20.8.1.6 <a href="#">Designing your own window-access marker names</a> on page 407. The keyword part of the marker name can be any of the following: <ul style="list-style-type: none"> <li>• <b>Size</b> (see <a href="#">Table 20-1</a>); for example, <b>JH2PopSize</b> or <b>JH2SecSize</b></li> <li>• <b>Name</b> or <b>Location</b>, for secondary windows only (see <a href="#">Table 20-1</a>): <b>JH2SecName</b>, <b>JH2SecLocation</b></li> <li>• the base name of a window-object access keyword listed in <a href="#">Table 20-2</a>; for example, <b>JH2PopText</b> or <b>JH2SecGraphic</b>.</li> </ul>
<i>Marker content</i>	The content of the marker is any value you could assign to the keyword in section <code>[JavaHelpOptions]</code> ; see §20.8.1.4 <a href="#">Specifying window-access object properties</a> on page 405. The marker content overrides the corresponding keyword setting, but only for the next pop-up or secondary window link in the file.
<i>Resize individual pop-up windows</i>	Use <b>JH2PopSize</b> 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 <b>JH2PopSize</b> with content <code>300,75</code> somewhere before the pop-up hotspot, for a 300-pixel-wide and 75-pixel-high pop-up window.

### 20.8.1.6 Designing your own window-access marker names

To specify JavaHelp 2 window-access marker-name prefixes other than **JH2Pop** and **JH2Sec** (see §20.8.1.5 [Overriding window-access properties with markers](#) on page 407):

```
[JavaHelpOptions]
; PopMarkerPrefix = prefix for pop-up window marker type,
; default JH2Pop
PopMarkerPrefix = JH2Pop
; SecMarkerPrefix = prefix for secondary window marker type
; default JH2Sec
SecMarkerPrefix = JH2Sec
```

<i>Prefix required</i>	The window-access marker names require <i>some</i> prefix; if you try to assign an empty prefix, <b>DITA2Go</b> 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 **DITA2Go** 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 DITA files, a change that cannot be accomplished with a template.

## 20.8.2 Specifying window parameters for Oracle Help

**DITA2Go** 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 20-3](#) describes the properties you can specify for each window.

**Table 20-3 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
buttons	Sum of the following hexadecimal values, in hexadecimal: <ul style="list-style-type: none"> <li>4 - Remove default buttons</li> <li>40 - Add URL display</li> <li>400 - Add Navigator button</li> <li>2000 - Print</li> <li>4000 - Back and Forward</li> <li>8000 - Search</li> <li>10000 - Dock and Undock</li> </ul>
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 §16.8 [Creating pop-up topics for Help systems](#) on page 263. 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.

## 20.8.3 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 §16.7 [Jumping to secondary windows in Help systems](#) on page 262. The window name `popup` is reserved for pop-up windows.

## 20.9 Linking to destinations within topics

After you convert a DITA 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
```

## 20.10 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). **DITA2Go** 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 **DITA2Go** 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 DITA document, see §16.6 [Providing related-topic links for Help systems](#) on page 258.

To create a “pool” of ALinks, where every instance serves both as a jump and a target, you can use a format combined with **DITA2Go** 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 **DITA2Go** code that automatically interprets the alink protocol (see §16.6.5.1 [Configuring ALink jumps](#) on page 261); what you build is passed through to Oracle Help unaltered.

## 20.11 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:

§16.11 [Setting up a dynamic modular Help system](#) on page 280

## 20.12 Setting up CSH for JavaHelp or Oracle Help

For context-sensitive help, you insert symbolic IDs into your DITA files as TopicAlias PI markers, at the appropriate topic start points. **DITA2Go** puts these IDs in the .jhm map file for you.

By default, **DITA2Go** removes punctuation and spaces from TopicAlias PI 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 §16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277.



# 21 Generating Eclipse Help

---

**DITA2Go** 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 22 and sections 27 through 43 apply also. Topics include:

- §21.1 [Understanding how Eclipse Help works](#) on page 413
- §21.2 [Setting up an Eclipse Help project](#) on page 413
- §21.3 [Configuring Eclipse Help manifest files](#) on page 416
- §21.4 [Configuring contents and index for Eclipse Help](#) on page 420
- §21.5 [Configuring search properties for Eclipse Help](#) on page 423
- §21.6 [Merging Eclipse Help projects](#) on page 423
- §21.7 [Setting up CSH for Eclipse Help](#) on page 425
- §21.8 [Packaging Eclipse Help files](#) on page 427

*See also:*

- §16 [Producing on-line Help](#) on page 243

## 21.1 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.

**DITA2Go** 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).

## 21.2 Setting up an Eclipse Help project

*In this section:*

- §21.2.1 [Creating an Eclipse Help project](#) on page 413
- §21.2.2 [Deciding where to locate configuration settings](#) on page 414
- §21.2.2 [Deciding where to locate configuration settings](#) on page 414
- §21.2.4 [Making sure links work in Eclipse Help](#) on page 415
- §21.2.5 [Disabling breadcrumb trails in Eclipse Help](#) on page 415

### 21.2.1 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 DITA document is located.

2. Copy configuration file `_d2eclipse.ini` from your **DITA2Go** `config` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:
3. Use a text editor to edit `_d2eclipse.ini` (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

## 21.2.2 Deciding where to locate configuration settings

When you set up an Eclipse Help project, if configuration file `_d2htmlhelp.ini` is not already present in the project directory, you must copy this file from your **DITA2Go** `config\local` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29).

*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>_d2eclipse.ini</code>	Current project directory
All Eclipse Help projects	<code>local_d2eclipse_config.ini</code>	<code>%omshome%\d2g\local\config\</code>

See §39.4 [Deciding which configuration file to edit](#) on page 734.

To determine which configuration settings will produce the appearance and functionality you want, also see:

- §22 [Converting to HTML/XHTML](#) on page 429
- §27 [Splitting and extracting files](#) on page 523
- §30 [Mapping text formats to HTML/XML](#) on page 565
- §32 [Including graphics in HTML](#) on page 611
- §33 [Converting tables to HTML](#) on page 625

## 21.2.3 Specifying Eclipse Help output options

To add or change any of the options described in this section, edit configuration file `_d2eclipse.ini`, located in the project directory.

*In this section:*

- §21.2.3.1 [Specifying a different output file extension](#) on page 414
- §21.2.3.2 [Specifying the target Eclipse version](#) on page 414
- §21.2.3.3 [Choosing whether to generate plugin.xml](#) on page 415

### 21.2.3.1 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
```

### 21.2.3.2 Specifying the target Eclipse version

By default, **DITA2Go** 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.

### 21.2.3.3 Choosing whether to generate plugin.xml

By default, **DITA2Go** 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 **DITA2Go** project.

To direct **DITA2Go** *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
```

## 21.2.4 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 **DITA2Go** to leave case alone in hypertext links, you can override this setting in a project or local configuration file.

See §28.2.6 [Forcing link text to lowercase](#) on page 549.

## 21.2.5 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 §21.4.3.3 [Enabling mid-topic links from the TOC](#) on page 422.

## 21.3 Configuring Eclipse Help manifest files

*In this section:*

- §21.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 416
- §21.3.2 [Specifying Eclipse Help plug-in properties](#) on page 416
- §21.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 417
- §21.3.4 [Configuring the plug-in manifest file for Eclipse Help](#) on page 418

### 21.3.1 Specifying a Java manifest file for Eclipse Help

By default, **DITA2Go** 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`, **DITA2Go** 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 §21.3.2 [Specifying Eclipse Help plug-in properties](#) on page 416.
- Includes a processing instruction in `plugin.xml`, and sets the plug-in schema version to 3.2 unless you specify a different version; see §21.3.4.2 [Including a processing instruction to validate plugin.xml](#) on page 418. 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 §44.6.1 [Specifying a wrap directory](#) on page 792. When `UseManifest=No`, subdirectory `META-INF` is not created.

### 21.3.2 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 §21.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 416):

**Table 21-1: 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
<a href="#">Title</a>	PluginName	Bundle-Name	name
<a href="#">ID</a>	PluginID	Bundle-SymbolicName	id
<a href="#">Provider name</a>	PluginProvider	Bundle-Vendor	provider-name
<a href="#">Product version</a>	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 = DITA2Go Eclipse Help
```

In `MANIFEST.MF`, this title becomes the `Bundle-Name` value. The default value is the base name of your `.bookmap` or `.ditamap` file.

*ID* To specify a plug-in ID:

```
[EclipseHelpOptions]
; PluginID = text used in <plugin> or manifest for id attribute
PluginID = com.dita2go.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.dita2go.help`.

*Provider name* To specify the plug-in provider:

```
[EclipseHelpOptions]
; PluginProvider = text used in <plugin> or manifest for provider-name
; attribute
PluginProvider = dita2go.com
```

In `MANIFEST.MF`, the provider name becomes the `Bundle-Vendor` value. The default value is `dita2go.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`.

### 21.3.3 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`. **DITA2Go** can create this file for you, and does so by default when `UseManifest=Yes`; see §21.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 416.

To prevent **DITA2Go** 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`, **DITA2Go** creates `MANIFEST.MF` in the project directory; if this file is already present, **DITA2Go** overwrites it.

When `WriteManifest=No`, **DITA2Go** creates `MANIFEST.MF` only if this file is not already present in the project directory; **DITA2Go** does not overwrite an existing `MANIFEST.MF`. This setting allows you to customize `MANIFEST.MF` outside of **DITA2Go**, 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`, **DITA2Go** adds “`; singleton:=true`” after the value of `Bundle-SymbolicName` in `MANIFEST.MF`.



## 21.3.4 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, **DITA2Go** creates this manifest file based on settings you provide. However, you can exclude creation of `plugin.xml` from your project; see §21.2.3.3 [Choosing whether to generate plugin.xml](#) on page 415.

*In this section:*

- §21.3.4.1 [Creating plugin.xml for Eclipse Help](#) on page 418
- §21.3.4.2 [Including a processing instruction to validate plugin.xml](#) on page 418
- §21.3.4.3 [Specifying the plug-in schema version for plugin.xml](#) on page 419
- §21.3.4.4 [Specifying Eclipse Help TOC properties in plugin.xml](#) on page 419
- §21.3.4.5 [Specifying Eclipse Help index properties in plugin.xml](#) on page 419
- §21.3.4.6 [Including a runtime element in plugin.xml](#) on page 419
- §21.3.4.7 [Including or excluding full-text search for Eclipse Help](#) on page 420
- §21.3.4.8 [Specifying Eclipse Help CSH properties in plugin.xml](#) on page 420

*See also:*

- §21.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 417

### 21.3.4.1 Creating plugin.xml for Eclipse Help

To direct **DITA2Go** 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`, **DITA2Go** creates `plugin.xml` in the project directory; if `plugin.xml` is already present, **DITA2Go** overwrites it.

When `WritePlugin=No`, **DITA2Go** creates `plugin.xml` only if this file is not already present in the project directory; **DITA2Go** does not overwrite an existing `plugin.xml`. This setting allows you to customize `plugin.xml` outside of **DITA2Go**, without losing your customizations during subsequent conversions.

### 21.3.4.2 Including a processing instruction to validate plugin.xml

By default, **DITA2Go** 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`, **DITA2Go** 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 §21.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 417. The value of the version attribute represents the plug-in schema version; see §21.3.4.3 [Specifying the plug-in schema version for plugin.xml](#) on page 419.

### 21.3.4.3 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 §21.3.4.2 [Including a processing instruction to validate plugin.xml](#) on page 418.

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 §21.3.3 [Configuring the Java manifest file for Eclipse Help](#) on page 417), 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 §21.2.2 [Deciding where to locate configuration settings](#) on page 414).

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.

### 21.3.4.4 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 §21.4.3 [Configuring contents properties for Eclipse Help](#) on page 421.

### 21.3.4.5 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 §21.4.4 [Configuring index properties for Eclipse Help](#) on page 423.

### 21.3.4.6 Including a runtime element in plugin.xml

By default, **DITA2Go** 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 **DITA2Go** include a `<runtime/>` element in `plugin.xml`:

```
[EclipseHelpOptions]
UseRuntime = Yes
```

### 21.3.4.7 Including or excluding full-text search for Eclipse Help

By default, **DITA2Go** 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
```

### 21.3.4.8 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** PI markers in your DITA document. Place an **EclipseContext** PI marker that contains a context ID name in each topic to be referenced by an infopop. Add the infopop description in the **DITA2Go** configuration file, in section `[EclipseHelpContexts]`.

See §21.7 [Setting up CSH for Eclipse Help](#) on page 425.

## 21.4 Configuring contents and index for Eclipse Help

*In this section:*

- §21.4.1 [Choosing contents and index methods for Eclipse Help](#) on page 420
- §21.4.2 [Supplying path information for contents and index links](#) on page 421
- §21.4.3 [Configuring contents properties for Eclipse Help](#) on page 421
- §21.4.4 [Configuring index properties for Eclipse Help](#) on page 423

### 21.4.1 Choosing contents and index methods for Eclipse Help

To direct **DITA2Go** 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), **DITA2Go** creates only `toc.xml`. Contents file `toc.xml` is required for Eclipse Help.

When `ListType=Both`, **DITA2Go** creates both `toc.xml` and `index.xml`. Index file `index.xml` is supported only in Eclipse version 3.2 and later versions.

See also:

- §16.3.1 [Modifying contents or index production for HTML-based Help](#) on page 249.
- §21.4.3 [Configuring contents properties for Eclipse Help](#) on page 421
- §21.4.4 [Configuring index properties for Eclipse Help](#) on page 423

## 21.4.2 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 §21.7 [Setting up CSH for Eclipse Help](#) on page 425):

```
[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.

## 21.4.3 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 **DITA2Go** generates will be the primary TOC and helpset.

Properties described in the following sections are included in `toc.xml`:

- §21.4.3.1 [Providing a title for the TOC](#) on page 421
- §21.4.3.2 [Specifying a starting topic](#) on page 421
- §21.4.3.3 [Enabling mid-topic links from the TOC](#) on page 422
- §21.4.3.4 [Directing TOC links to top of topic page](#) on page 422

Additional TOC properties are specified in the plug-in manifest; see §21.3.4.4 [Specifying Eclipse Help TOC properties in plugin.xml](#) on page 419. See also §16.4 [Configuring contents entries for Help systems](#) on page 250.

### 21.4.3.1 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 §21.3.1 [Specifying a Java manifest file for Eclipse Help](#) on page 416. This is not usually what you want.

### 21.4.3.2 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`, **DITA2Go** 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, **DITA2Go** 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
```

### 21.4.3.3 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 **DITA2Go** 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), **DITA2Go** omits anchors from TOC entries, enabling Eclipse Help native breadcrumbs but disabling mid-topic jumps from the TOC.

When `TocNamesFileOnly=No`, **DITA2Go** includes anchors in TOC entries, enabling mid-topic jumps from the TOC, but disabling Eclipse Help native breadcrumbs.

See also:

§21.2.5 [Disabling breadcrumb trails in Eclipse Help](#) on page 415

### 21.4.3.4 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 §28.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 549.

### 21.4.4 Configuring index properties for Eclipse Help

Only Eclipse version 3.2 and later versions support an index. **DITA2Go** produces the index from your DITA index entries, in the form of XML file `index.xml` with links to your HTML topic files. When `EclipseVer=2` or later (see §21.2.2 [Deciding where to locate configuration settings](#) on page 414), **DITA2Go** 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 §21.3.4.5 [Specifying Eclipse Help index properties in plugin.xml](#) on page 419. See also §16.5 [Configuring index entries for Help systems](#) on page 251.

## 21.5 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 DITA PI marker of type **MetaDescription**, or use the content of a dedicated paragraph format; see §22.4.6 [Supplying content for the `<meta>` tag](#) on page 436.

**Note:** In Eclipse 3.4 this technique works only for HTML files, not XHTML files.

## 21.6 Merging Eclipse Help projects

If your customers are using Eclipse version 3.4 or later, you can simply insert PI marker in DITA XML 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 §21.6.1 [Linking primary content to secondary TOCs](#) on page 423.

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 §21.6.2 [Linking secondary TOCs to primary content \(deprecated\)](#) on page 424. For versions of Eclipse starting with Eclipse 3.4, this method is deprecated.

*In this section:*

§21.6.1 [Linking primary content to secondary TOCs](#) on page 423

§21.6.2 [Linking secondary TOCs to primary content \(deprecated\)](#) on page 424

### 21.6.1 Linking primary content to secondary TOCs

To link a secondary helpset into your primary Eclipse Help system, insert a PI marker of type **EclipseLink** in the DITA 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 §16.4.3 [Setting contents levels for HTML-based Help](#) on page 251.

In Eclipse Help output, **DITA2Go** generates the following from each **EclipseLink** PI 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 §21.6.2 [Linking secondary TOCs to primary content \(deprecated\)](#) on page 424.

## 21.6.2 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 §21.6.1 [Linking primary content to secondary TOCs](#) on page 423 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 DITA document insert an **EclipseAnchor** PI marker in the paragraph that immediately precedes the point where you want the secondary helpset linked.

You can use a PI marker either of type **EclipseAnchor** or of type **EclipseLink**. Marker content consists of the following:

<b>EclipseAnchor</b>	TOC level number, space, anchor name.
<b>EclipseLink</b>	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 §16.4.3 [Setting contents levels for HTML-based Help](#) on page 251. 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.

## 21.7 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:*

- §21.7.1 [Understanding how DITA2Go generates context links](#) on page 425
- §21.7.2 [Naming context file and attribute for secondary plug-ins](#) on page 426
- §21.7.3 [Configuring context IDs and context anchors](#) on page 426
- §21.7.4 [Configuring context descriptions](#) on page 426
- §21.7.5 [Locating context information](#) on page 427

*See also:*

- §16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277

### 21.7.1 Understanding how DITA2Go generates context links

**DITA2Go** recognizes **EclipseContext** PI 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 §16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277. 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 §21.2.2 [Deciding where to locate configuration settings](#) on page 414.

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>

```

**DITA2Go** creates contexts.xml afresh every time you run the conversion, unless you say not to; see §21.3.4.8 [Specifying Eclipse Help CSH properties in plugin.xml](#) on page 420.

## 21.7.2 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.

## 21.7.3 Configuring context IDs and context anchors

To provide entry points for infopop calls from an application to an Eclipse Help system, insert **EclipseContext** PI markers in target topics in your DITA XML file. The content of an **EclipseContext** PI marker is the context ID for the infopop, which becomes a <context> element in toc.xml.

By default, **DITA2Go** inserts each context anchor at the beginning of a topic, regardless of where in the topic you place the **EclipseContext** PI marker. To produce mid-topic entry points instead:

```

[EclipseHelpOptions]
; ContextAnchors = No (default, filename only) or Yes (refer to para)
ContextAnchors = Yes

```

When ContextAnchors=Yes, **DITA2Go** inserts a context anchor at the beginning of the paragraph that contains an **EclipseContext** PI marker.

When ContextAnchors=No, the context anchor appears at the beginning of the topic.

## 21.7.4 Configuring context descriptions

By default, **DITA2Go** 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 **DITA2Go** 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
```

### 21.7.5 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 §21.4.2 [Supplying path information for contents and index links](#) on page 421.

## 21.8 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.

If you run **DITA2Go** in a `.bat` file, you can add a command to create `doc.zip`. For example, if you are using the command-line version of WinZip:

```
"c:\program files\winzip\wzip" -a doc.zip -xg:\*.xml g:\*.*
```

This command adds everything in the current directory to `doc.zip`, except files with extension `.xml`.

**DITA2Go** 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.

*In this section:*

§21.8.1 [Specifying a ZIP command for doc.zip](#) on page 427

§21.8.2 [Specifying ZIP command parameters](#) on page 427

§21.8.3 [Archiving Eclipse Help files](#) on page 428

*See also:*

§16.2.2 [Compiling and distributing Help systems](#) on page 247

§44.6 [Assembling files for distribution](#) on page 792

### 21.8.1 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"
```

### 21.8.2 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 §44.2 [Activating and logging production of deliverables](#) on page 788), 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 §21.8.3 [Archiving Eclipse Help files](#) on page 428.

### 21.8.3 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 **DITA2Go** automation settings to archive everything needed for your Eclipse Help plug-in: XML files along with HTML files or `doc.zip`. See §44.11 [Archiving deliverables](#) on page 803. 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.

# 22 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:

- §22.1 [Deciding which type of output to produce](#) on page 430
- §22.2 [Setting up an HTML project](#) on page 430
- §22.3 [Including starting code and entity references](#) on page 432
- §22.4 [Supplying values for the <head> element](#) on page 432
- §22.5 [Specifying HTML <body> attributes](#) on page 438
- §22.6 [Specifying document-wide properties for HTML](#) on page 438
- §22.7 [Defining text colors for HTML](#) on page 439
- §22.8 [Importing HTML files as insets](#) on page 441
- §22.9 [Providing hover text for links in HTML](#) on page 441
- §22.10 [Generating XHTML for Confluence 4.x](#) on page 442
- §22.11 [Exporting content for database input](#) on page 443
- §22.12 [Specifying a starting topic for HTML or XHTML](#) on page 443
- §22.13 [Using framesets](#) on page 443
- §22.14 [Passing W3C validation tests](#) on page 445

*See also:*

<i>HTML-based help</i>	§16 <a href="#">Producing on-line Help</a> on page 243, if you plan to generate a Help system.
<i>Generic XML</i>	§23 <a href="#">Converting to generic XML</a> on page 449, for settings specific to XML (but <i>not</i> DITA or DocBook XML).
<i>DITA XML</i>	§24 <a href="#">Converting to DITA XML</a> on page 455.
<i>DocBook XML</i>	§26 <a href="#">Converting to DocBook XML</a> on page 499.
<i>File splitting</i>	§27 <a href="#">Splitting and extracting files</a> on page 523 for settings that govern the subdivision of a DITA document into HTML topic files.
<i>Links</i>	§28 <a href="#">Creating HTML links</a> on page 545 and §29 <a href="#">Providing navigation in HTML</a> on page 555 for ways to create navigation links.
<i>Formats</i>	§30 <a href="#">Mapping text formats to HTML/XML</a> on page 565 for settings that map paragraph and character formats to HTML elements, and that position graphics and equations.
<i>CSS</i>	§31 <a href="#">Setting up CSS for HTML</a> on page 591 if you plan to use CSS.
<i>Graphics</i>	§32 <a href="#">Including graphics in HTML</a> on page 611 for ways to convert graphics and equations, and specify image properties.
<i>Tables</i>	§33 <a href="#">Converting tables to HTML</a> on page 625 for ways to specify table structure and display properties.
<i>WAI markup</i>	§34 <a href="#">Generating WAI markup for HTML</a> on page 649 for ways to add WAI (Web Accessibility Initiative) attributes to tables, images, and links.
<i>Macros</i>	§37 <a href="#">Working with macros</a> on page 679 for ways to use <b>DITA2Go</b> macros.
<i>Markers for markup</i>	§38 <a href="#">Working with processing instructions</a> on page 717 for ways to use DITA PI markers to include HTML code and <b>DITA2Go</b> directives in your DITA document.

## 22.1 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 §22.4.1 [Specifying HTML/XML version, DOCTYPE, and DTD](#) on page 432.

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 **DITA2Go** XHTML output as input to Calibre, which is free:

<http://calibre-ebook.com/>

*Internet browsers* If your output will be displayed on the Web, consider the differences among browsers. If you use CSS (see §31 [Setting up CSS for HTML](#) on page 591), 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.

## 22.2 Setting up an HTML project

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	_d2html_config.ini
XHTML	_d2xhtml_config.ini

Or, to apply the changes to all of your HTML (or XHTML) projects, edit the corresponding configuration template:

%omsyshome%\d2g\local\config\local\_d2\*ml\_config.ini

See §39.4 [Deciding which configuration file to edit](#) on page 734.

*In this section:*

§22.2.1 [Creating an HTML or XHTML project](#) on page 430

§22.2.2 [Specifying a different output file extension](#) on page 431

§22.2.3 [Checking HTML or XML output files for DITA2Go version](#) on page 431

§22.2.4 [Using XHTML tagging rules for HTML](#) on page 431

### 22.2.1 Creating an HTML or XHTML project

*Easier:* use the **DITA2Go** Project Manager to start a new project; see §2.1 [Creating a DITA2Go conversion project](#) on page 39.

To create an HTML or XHTML project without using the **DITA2Go** Project Manager:

1. Create a directory for output files, separate from the directory where your DITA document is located.
2. Copy the appropriate configuration file `_d2*ml.ini` from `%OMSYSHOME%\d2g\system\starts`, or from an existing **DITA2Go**-to-HTML project, to your newly created project directory.
3. Use a text editor to edit the configuration file (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

### 22.2.2 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:

§24.2.2 [Specifying DITA output options](#) on page 456

§26.2.2.1 [Changing the DocBook output file extension](#) on page 501

### 22.2.3 Checking HTML or XML output files for DITA2Go version

If you recently installed a **DITA2Go** upgrade or beta version, after you run **DITA2Go**, 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 **DITA2Go** `drxml.dll` and `dwhtm.dll` components that were used to create the HTML file. See §A.1.5 [Check your version of DITA2Go](#) on page 820.

### 22.2.4 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 `<br />`).

All current HTML browsers accept these rules.

When you specify XHTML as your output type, in addition to a name attribute for anchors, **DITA2Go** 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.

## 22.3 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 §37 [Working with macros](#) on page 679.

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.

## 22.4 Supplying values for the <head> element

**DITA2Go** 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:*

- §22.4.1 [Specifying HTML/XML version, DOCTYPE, and DTD](#) on page 432
- §22.4.2 [Specifying namespace and language](#) on page 433
- §22.4.3 [Specifying character encoding for HTML](#) on page 434
- §22.4.4 [Including or omitting HTML/XML generator information](#) on page 435
- §22.4.5 [Specifying page titles for HTML output files](#) on page 435
- §22.4.6 [Supplying content for the <meta> tag](#) on page 436
- §22.4.7 [Specifying nonstandard values for declarations](#) on page 437

### 22.4.1 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=

```

**DITA2Go** 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, **DITA2Go** automatically sets `[HTMLOptions]HTMLVersion=3`, unless you override this setting in the configuration file.

*XHTML* When you specify XHTML as the output type, **DITA2Go** 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/xhtml1-transitional.dtd"

```

*DITA XML* When you specify DITA XML as the output type, **DITA2Go** 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 §24.3 [Specifying general options for DITA](#) on page 458.

*DocBook XML* When you specify DocBook XML as the output type, **DITA2Go** 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"

```

## 22.4.2 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 **DITA2Go** 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 §22.14 [Passing W3C validation tests](#) on page 445.

### 22.4.3 Specifying character encoding for HTML

HTML is based on Unicode. **DITA2Go** 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.

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 **DITA2Go** 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, **DITA2Go** does not actually produce any characters with values greater than 127 using the UTF-8 encoding; instead, **DITA2Go** uses entities for such characters, readable under any eight-bit encoding scheme.

For XHTML, you can specify a value for `XMLEncoding` (see §23.2.3 [Specifying character encoding for generic XML](#) on page 450) 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 **DITA2Go** 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:

§23.2.3 [Specifying character encoding for generic XML](#) on page 450

## 22.4.4 Including or omitting HTML/XML generator information

The header of each HTML and XML file generated by **DITA2Go** contains an element identifying the **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** 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.

## 22.4.5 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 DITA PI 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 §27.5.2 [Specifying page titles for split or extract files](#) on page 531.

**Note:** If some of your output files show *Test File from DITA2Go* 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 DITA2Go
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 DITA source file as the page title for each HTML file derived from that DITA file:

```
[HTMLOptions]
Title=<$$_basefile>
```

See §27.5.2.6 [Assigning a default title](#) on page 533.

*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 §27.5.2.2 [Assigning a title with a paragraph format](#) on page 532.

*Use a DITA PI marker*

You can specify an individual page title with a DITA **Title** PI 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 §27.5.2.5 [Assigning a title with a marker](#) on page 533.

*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 **DITA2Go**, not to any replacement name you may have specified for a split or extract file. See §27.5.2.4 [Assigning a title with a file name](#) on page 533.

## 22.4.6 Supplying content for the <meta> tag

**DITA2Go** supports several ways to provide content for <meta> tags in the <head> element of each HTML page.

*In this section:*

§22.4.6.1 [Providing meta content with paragraph formats](#) on page 436

§22.4.6.2 [Providing meta content with DITA PI markers](#) on page 437

§22.4.6.3 [Providing meta content with DITA2Go macros](#) on page 437

### 22.4.6.1 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 §30.2.6 [Eliminating unwanted paragraphs](#) on page 569.

### 22.4.6.2 Providing meta content with DITA PI markers

- If you give a custom PI marker type a name that starts with **Meta**, **DITA2Go** 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.

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.

**DITA2Go** concatenates all markers for the same attribute in the same HTML file. You can just add more PI markers of the same type, and continue the content. **Meta\*** PI markers are concatenated into one <meta> tag within the same split or extract file, but not across file boundaries.

### 22.4.6.3 Providing meta content with DITA2Go macros

You can put <meta> tags directly into the configuration file as macros, and not have them in your DITA document at all. See §27.6.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 535 and §37.1 [Defining and invoking macros](#) on page 679.

## 22.4.7 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](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
```

## 22.5 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 §33.4.1 [Specifying attributes for all tables](#) on page 632.

## 22.6 Specifying document-wide properties for HTML

*In this section:*

§22.6.1 [Specifying a default DPI setting](#) on page 438

§22.6.2 [Suppressing closing </p> tags for HTML](#) on page 438

§22.6.3 [Suppressing line breaks in HTML and XML output](#) on page 439

§22.6.4 [Preventing adjacent <pre> elements from merging](#) on page 439

### 22.6.1 Specifying a default DPI setting

To convert from other sizes to pixels, for such purposes as table-column sizing, indenting tables, and indenting graphics, **DITA2Go** uses this DPI setting:

```
[HTMLOptions]
; ConversionDPI = 96 (default), used when converting sizes to pixels
ConversionDPI = 96
```

### 22.6.2 Suppressing closing </p> tags for HTML

By default, **DITA2Go** provides closing tags `</tagname>`, to conform to W3C validation requirements for XHTML. (One exception: unless you specify XHTML as the output type, **DITA2Go** does not generate closing tags for list items; see §30.11.2 [Converting list formats to HTML list styles](#) on page 585.)

To eliminate </p> closing tags:

```
[HTMLOptions]
; NoParaClose = No (default) or Yes (suppress </p> closing tags)
NoParaClose = Yes
```

### 22.6.3 Suppressing line breaks in HTML and XML output

By default, **DITA2Go** 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 **DITA2Go** 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 §23.2.5 [Preventing arbitrary line breaks in XML text elements](#) on page 451.

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 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 §30.2.4 [Stripping paragraph properties](#) on page 568.

### 22.6.4 Preventing adjacent <pre> elements from merging

By default, for HTML output **DITA2Go** 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.

## 22.7 Defining text colors for HTML

The best way to adjust text colors for HTML output is to assign colors to output formats; see:

§7.6.5 [Specifying inline properties for paragraph and character formats](#) on page 123

§7.6.6 [Specifying block properties for paragraph formats](#) on page 124

§8.3 [Defining shading format components](#) on page 145

Those methods use CSS. Consider using the method described here only if you cannot use CSS.

*In this section:*

§22.7.1 [Numbering and defining text colors](#) on page 440

§22.7.2 [Using Web-safe colors](#) on page 440

## 22.7.1 Numbering and defining text colors

You can assign hexadecimal RGB color values to arbitrary index numbers, and then use those index numbers in certain configuration settings to represent color values.

To number colors and specify color values:

```
[Colors]
; color number 1-254 = hex RGB value; color number 0 is invisible
; 1..8 = black, white, red, green, blue, cyan, magenta, yellow
; 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

To map a reserved color to some other color, assign its number a new RGB value; for example:

```
[Colors]
; Replace Cyan with SeaWater
6 = 0099CC
```

See also §30.8 [Specifying text colors for HTML](#) on page 580.

To define a new color, assign a hexadecimal RGB value to any decimal integer from 9 through 254 (0 through 8 are reserved). For example:

```
[Colors]
; Use deep pink for table headings
105 = FF3399
```

## 22.7.2 Using Web-safe colors

It is best to use Web-safe colors for HTML; otherwise you could run into browser palette issues.

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. For example, RGB hexadecimal value 0099CC yields the color shown in [Figure 22-1](#).

**Figure 22-1 RGB color 0099CC**



[Table 22-1](#) lists the values you can use to define Web-safe RGB colors, in three different units of measurement.



**Table 22-1 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; <b>DITA2Go</b> [Colors]
Decimal	0	51	102	153	204	255	Windows, some applications

## 22.8 Importing HTML files as insets

To include existing HTML code as an inset, by importing HTML from a source other than the DITA files you are converting, use a DITA **HTML Macro** PI marker. The content of the marker should look like this:

```
<?dthtm HTML Macro="<$.\filename.htm" ?>
```

where *filename.htm* is the name of the HTML file you want to import. Place the marker in your DITA document wherever you want the imported HTML to appear in your **DITA2Go** 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 **DITA2Go** macro expression (see §37.6 [Using expressions in macros](#) on page 700) in the content of the **HTML Macro** marker. Code such as the following would select all text between `<body>` and `</body>` from *filename.htm*:

```
<?dthtm HTML Macro='<$(($.\\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 **DITA2Go** expression evaluator does not take it as a division operator; and two of them, the first to escape the second backslash in the PI marker.

## 22.9 Providing hover text for links in HTML

Hover text in HTML is produced from the value of the HTML `title` attribute of a tag (usually a `<span>` tag) that encloses the term. The attribute value may not contain a carriage return, nor the symbols `<`, `>`, `"`, `'`, or `&`.

**DITA2Go** provides mouseover hover text for all cross references and links to topics in your DITA source, including those added by `@keyref`, internal scope only:

- For links to topics the hover text is the content of the `<shortdesc>` or `<abstract>` element; if neither is present, the link has no hover text.
- For `<glossentry>` topics, the hover text is the content of the `<glossdef>` element; links are still present and active, if (for example) a user wants more detail.
- For a cross reference to a section or figure or table, the hover text is the title, and any autonumber would be included also.

If an element has an `@href` attribute either because of the element type (such as `<xref>`) or because it has a `@keyref` to a `<keydef>` with an `@href`, **DITA2Go** can use the reference for HTML `title` attribute text. The exceptions are:

- external and peer scope, where **DITA2Go** cannot access the other end of the `@href`



- references to elements that lack titles, where the text to use is not necessarily clear.

No special settings are needed to enable hover text for DITA links and cross references.

## 22.10 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, **DITA2Go** provides a way to produce the required markup. See:

<https://confluence.atlassian.com/display/DOC/Confluence+Storage+Format>

To direct **DITA2Go** to produce XHTML for Confluence 4.x:

```
[HTMLOptions]
; Confluence = No (default, use normal linking)
; or Yes (make Confluence links)
Confluence = Yes
```

When Confluence=Yes, **DITA2Go** automatically sets the options listed in [Table 22-2](#). You can override these individually if necessary.

**Table 22-2 Default options for Confluence 4.x XHTML**

Configuration section	Setting	Value	Reference
[HTMLOptions]	ConfluenceLinks	Yes	<a href="#">22.10</a>
	RemoveANames	Yes	<a href="#">23.4</a>
	NoLocations	Yes	<a href="#">28.3.2</a>
	NoFonts	Yes	<a href="#">30.6.3</a>
	UseHash	No	<a href="#">23.4</a>
	AlignAttributes	No	<a href="#">30.4</a>
	UseXMLDeclaration	No	<a href="#">22.4.7</a>
	UseDOCTYPE	No	<a href="#">22.4.1</a>
	UseHeadAndBody	No	<a href="#">22.4.7</a>
[CSS]	UseCSS	No	<a href="#">31.4.1</a>
	UseSpanAsDefault	No	<a href="#">31.7.3</a>

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 **DITA2Go** must be imported into Confluence one at a time. As of this writing, no batch import utility is available.

## 22.11 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
```

## 22.12 Specifying a starting topic for HTML or XHTML

To specify which topic file should open first in a browser when you click **View Output** in the **DITA2Go** Project Manager:

```
[HTMLOptions]
; ViewOutputFile = file in wrap directory for Project Manager to open
; in browser upon View Output.
ViewOutputFile = filename.ext
```

The file you specify must be present in the wrap directory; see §44.6.1 [Specifying a wrap directory](#) on page 792. Or, if you include a path, the path must be relative to the wrap directory.

By default, the Project Manager uses the value of `[Contents]TOCFile`, and looks for a file of that name in the wrap directory (see §14.3.1 [Specifying a file name and title for the TOC](#) on page 199). If `TOCFile` is not set, or the named file is not present in the wrap directory, the Project Manager looks in the wrap directory for a file that has the map name for your project and a `FileSuffix` extension (see §4.1.6 [Checking output type and file extension](#) on page 70).

The Project Manager opens the starting topic file in the default browser. To specify a different browser or viewer:

```
[HTMLOptions]
; ViewOutputCommand = path\to\browser, default none
```

See [View-output command](#) under §1.3.7 [Establish system-wide configuration settings](#) on page 33.

## 22.13 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 **DITA2Go** macro (see §37 [Working with macros](#) on page 679), 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 (`d2h_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 PI 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, **DITA2Go** 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 `<a href="..." target="_top">`, 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 §28.4 [Creating jumps to particular windows for HTML](#) on page 550.

**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.

## 22.14 Passing W3C validation tests

**DITA2Go** generates W3C-valid code, and with the appropriate settings is completely conformant.

To check the validity of your HTML output:

<http://validator.w3.org/>

*In this section:*

§22.14.1 [Understanding limitations of W3C validation](#) on page 445

§22.14.2 [Replacing high ASCII characters for W3C validation](#) on page 445

§22.14.3 [Eliminating <nobr> tags](#) on page 447

§22.14.4 [Avoiding redundant attribute assignments in tables](#) on page 447

### 22.14.1 Understanding limitations of W3C validation

**DITA2Go** 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 DITA 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 DITA document and to your configuration settings.

### 22.14.2 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 **DITA2Go** 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 `<ul>` tags) become mid-dots
- all other characters in the range are dropped, unless you map them yourself; see §30.4 [Assigning properties to text formats](#) on page 570.

Table 22-3 shows how **DITA2Go** treats characters in this range when `ValidOnly=Yes`. Depending on which version of the *DITA2Go User's Guide* you are using to view the table, some characters might not be displayed.

**Table 22-3 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)
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 <code>&lt;ul&gt;</code> 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>

**Table 22-3 Characters replaced or removed for W3C validation (continued)**

Value	Character	Name	Replacement character (if any)
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:

§22.4.3 [Specifying character encoding for HTML](#) on page 434

§23.2.3 [Specifying character encoding for generic XML](#) on page 450

§30.4 [Assigning properties to text formats](#) on page 570

### 22.14.3 Eliminating <nobr> tags

By default, **DITA2Go** 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 <nobr> tags around nonbreaking
; hyphens, supported properly by all browsers),
; or No (required for W3C validation)
AllowNobr = No
```

### 22.14.4 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 §35.1.3.2 [Applying the scope method to all tables](#) on page 658):

```
[Tables]
AccessMethod=Scope
```

Then if you happen to include a **CellScope** PI marker in some table (see §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 6660, the `scope` attribute assignment appears twice in the output for that table cell.





# 23 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:*

§24 [Converting to DITA XML](#) on page 455

§26 [Converting to DocBook XML](#) on page 499

Topics for generic XML include:

§23.1 [Setting up a generic XML project](#) on page 449

§23.2 [Specifying generic XML output settings](#) on page 449

§23.3 [Providing XML tags and structure](#) on page 451

§23.4 [Configuring links for generic XML](#) on page 454

Check the *W3C Extensible Markup Language (XML) 1.0 (Second Edition)* recommendation for information about XML:

<http://www.w3.org/TR/REC-xml>

## 23.1 Setting up a generic XML project

For the most part **DITA2Go** conversions to XML employ the same project set-up options, conversion methods, macros, and configuration settings as conversions to HTML or XHTML; see §22.2 [Setting up an HTML project](#) on page 430.

*Conversion files* The same conversion files are generated and named the same way for XML as for HTML or XHTML.

*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
	Footnotes	Inline	Jump
	NoFonts	Yes	No
	UseAnums	No	Yes (except lists)
	UseFootXrefTag	Yes	No
	UseHeadAndBody	No	Yes
	XMLRoot	doc	html
	UseCALSMoDel	Yes	No
[Tables]	UseCALSMoDel	Yes	No

## 23.2 Specifying generic XML output settings

To add or change any of the options described in this section, edit configuration file `_d2xml.ini`, located in the project directory.

*In this section:*

§23.2.2 [Changing output XML version or file extension](#) on page 450

§23.2.3 [Specifying character encoding for generic XML](#) on page 450

§23.2.4 [Specifying the root element and content type](#) on page 451

§23.2.5 [Preventing arbitrary line breaks in XML text elements](#) on page 451

§23.2.6 [Specifying a starting topic for generic XML](#) on page 451

## 23.2.1 Creating a generic XML project

*Easier:* use the **DITA2Go** Project Manager to start a new project; see §2.1 [Creating a DITA2Go conversion project](#) on page 39.

To create a generic XML project without using the **DITA2Go** Project Manager:

1. Create a directory for output files, separate from the directory where your DITA document is located.
2. Copy configuration file `d2xml.ini` from your **DITA2Go** `config` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:
3. Use a text editor to edit `d2xml.ini` (see §3.1 [Working with DITA2Go configuration files](#) on page 49).

## 23.2.2 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`.

## 23.2.3 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. 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 **DITA2Go** 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, **DITA2Go** does not actually produce any characters with values greater than 127 using the UTF-8 encoding; instead, **DITA2Go** 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:*

§22.3 [Including starting code and entity references](#) on page 432

§22.4.3 [Specifying character encoding for HTML](#) on page 434

### 23.2.4 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](http://www.w3.org/Protocols/rfc1341/4_Content-Type.html)

### 23.2.5 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 **DITA2Go** 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 §22.6.3 [Suppressing line breaks in HTML and XML output](#) on page 439.

### 23.2.6 Specifying a starting topic for generic XML

To specify which topic file should open first in a viewer when you click **View Output** in the **DITA2Go** Project Manager, and which viewer or editor should be used, see §22.12 [Specifying a starting topic for HTML or XHTML](#) on page 443.

## 23.3 Providing XML tags and structure

*In this section:*

§23.3.1 [Deriving XML tags from format and class names](#) on page 452

§23.3.2 [Eliminating HTML attributes and tags for generic XML](#) on page 452

§23.3.3 [Including or excluding autonumbers](#) on page 453

*See also:*

§37.10 [Using macros to fine-tune HTML or XML output](#) on page 713

### 23.3.1 Deriving XML tags from format and class names

To aid in mapping formats to elements for XML output from an unstructured document, by default **DITA2Go** 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 an `outputclass` attribute
  - in CSS
  - in the `[ParaClasses]` or `[CharClasses]` section
  - in any other configuration-file section.

To produce valid XML, **DITA2Go** converts all tags to valid CSS names, without spaces, non-alphanumeric characters, leading digits, or accented characters (which become unaccented).

*Paragraph and character tags and attributes*

**DITA2Go** uses any tags and attributes you assign in configuration sections `[ParaTags]` and `[CharTags]`; see §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566. To apply an attribute to an individual paragraph or character span, insert an attribute PI marker in the instance; see §38.1 [Understanding DITA2Go PI markers](#) on page 717.

You can specify which names to use for XML tags in any or all of the following ways:

[Map class names to XML tags](#)

[Map format names to classes](#)

*Map class names to XML tags*

To map all CSS class names to XML tags (the default for XML output):

```
[CSS]
; ClassIsTag = No (default for HTML/XHTML)
; or Yes (default for Generic XML)
ClassIsTag=Yes
```

When `ClassIsTag=Yes`, any class names you assigned to formats in the `[ParaTags]` and `[CharTags]` sections become XML tags; see §31.5 [Understanding how CSS affects other options](#) on page 596. If `ClassIsTag=Yes`, also specify `[CSS]WriteClassAttributes=No`; see §31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593.

*Map format names to classes*

To explicitly map individual format names to class names:

```
[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.
```

*Specify all margins in CSS*

The following setting causes CSS entries to explicitly include all four margin values, even if some are zero:

```
[CSS]
; ZeroCSSMargins = No (default)
; or Yes (specify CSS margins even if zero)
ZeroCSSMargins=Yes
```

### 23.3.2 Eliminating HTML attributes and tags for generic XML

You can use configuration settings to eliminate the following HTML tags and attributes:

[Paragraph tags](#)

[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 §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566
- Use the NoPara property in [HTMLParaStyles]; see §30.2.4 [Stripping paragraph properties](#) on page 568.

*Table attributes*

By default, when you specify XML as the output type, **DITA2Go** refrains from automatically generating HTML table and cell attributes (see §33.4.1 [Specifying attributes for all tables](#) on page 632), while preserving any attributes you add specifically for XML in the configuration file or in PI markers:

```
[Tables]
; TableAttributes = Yes (HTML default, to allow automatically
; generated border, cellpadding, cellspacing, or No (XML default,
; to exclude those while keeping any attributes explicitly added
; in the .ini or in markers)
TableAttributes=No
```

By default, **DITA2Go** places `&nbsp;` in each table cell that would otherwise be empty (either because the cell contained only an empty paragraph, 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 `&nbsp;`. Or, you can specify any other content; see §33.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 640. However, a better approach would be to define common entities such as `&nbsp;` in your XML DTD; see §22.3 [Including starting code and entity references](#) on page 432.

*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
```

### 23.3.3 Including or excluding autonumbers

By default, **DITA2Go** 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 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:

§30.4 [Assigning properties to text formats](#) on page 570.

## 23.4 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 §26.3.2 [Configuring links for DocBook XML](#) on page 502. To configure links for DITA XML, see §24.9 [Configuring cross references and links for DITA](#) on page 489.

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
```

See also:

§24.9 [Configuring cross references and links for DITA](#) on page 489

§26.3.2 [Configuring links for DocBook XML](#) on page 502

§28.2.6 [Forcing link text to lowercase](#) on page 549

# 24 Converting to DITA XML

---

**DITA2Go** generates topics and maps for DITA XML output from DITA XML input. This section shows how to configure DITA-specific options. Topics include:

- §24.1 [Generating DITA XML output from DITA input](#) on page 455
- §24.2 [Setting up a DITA XML project](#) on page 455
- §24.3 [Specifying general options for DITA](#) on page 458
- §24.4 [Configuring DITA elements](#) on page 459
- §24.5 [Nesting DITA block elements](#) on page 471
- §24.6 [Specifying options for tables in DITA XML](#) on page 480
- §24.7 [Specifying options for images in DITA XML](#) on page 482
- §24.8 [Configuring DITA topics](#) on page 484
- §24.9 [Configuring cross references and links for DITA](#) on page 489
- §24.10 [Including CSH targets in DITA XML](#) on page 491
- §24.11 [Overriding DITA settings with markers](#) on page 491

*See also:*

- §25 [Configuring DITA maps](#) on page 493
- §41 [Working with content models](#) on page 753

## 24.1 Generating DITA XML output from DITA input

Why would you want to do such a thing? Perhaps to reorganize elements, move to a different DTD, adapt content or restructure for a different purpose?

To transform or repurpose DITA XML with **DITA2Go** requires two steps:

1. Map DITA input elements to formats.
2. Map those formats to DITA output elements.

You can name the formats directly after elements, either input or output elements, if that makes sense for your conversion. Conserving elements that do not require reworking is very simple: map unaffected elements to formats of the same name. However, you also have the flexibility to do much more when a clear element-to-element mapping will not suffice.

The only other way to rework DITA XML is to use XSLT, which for anything non-trivial requires serious programming skills.

## 24.2 Setting up a DITA XML project

To add or change any of the options described in this section, edit configuration file `_d2dita.ini`, located in your project directory. Or, to apply the changes to all of your DITA XML projects, edit the configuration file referenced by `_d2dita.ini`:

`%omshome%\d2g\local\config\local_d2dita_config.ini`.

See §39.4 [Deciding which configuration file to edit](#) on page 734.

*In this section:*

- §24.2.1 [Creating a DITA XML project](#) on page 456
- §24.2.2 [Specifying DITA output options](#) on page 456



§24.2.3 [Specifying DITA version](#) on page 456

§24.2.4 [Configuring the DITA DTD SYSTEM identifier](#) on page 457

## 24.2.1 Creating a DITA XML project

*Easier:* use the **DITA2Go** Project Manager to start a new project; see §2.1 [Creating a DITA2Go conversion project](#) on page 39.

To create a DITA XML project without using the **DITA2Go** Project Manager:

1. Create a directory for DITA output, separate from the directory where your DITA input document is located.
2. Copy configuration file `d2dita.ini` from your **DITA2Go** `config` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:

**DITA2Go** copies a new project configuration file, `_d2dita.ini`, to your project directory. 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 `_d2dita.ini`; see §3.1 [Working with DITA2Go configuration files](#) on page 49.

## 24.2.2 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.

## 24.2.3 Specifying DITA version

By default, **DITA2Go** 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 §4.1 [Working with content models](#) on page 753.

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.

## 24.2.4 Configuring the DITA DTD SYSTEM identifier

To have **DITA2Go** 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
```

## 24.2.5 Substituting document format names for default names

When **DITA2Go** creates a configuration file for a new DITA project, the default settings in all configuration sections that involve assigning a property to a format are based on the format names in a standard template.

If you want format names different from those in the standard template, *you must change these settings*. You must also add settings for other formats that you want to 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 template:

```
[DITAParaTags]
[DITACHarTags]
[DITACHarTypographics]
[DITALevels]
[DITAParents]
```

[DITAParaTags] Only a few paragraph formats are automatically mapped to DITA elements:

```
[DITAParaTags]
; para format (wildcards OK) = DITA element
Body=p
Heading*=title
Numbered=li
Numbered1=li
Bulleted=li
FigureTitle=title
CellBody=p
CellHeading=p
```

See §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460.

[DITACHarTags] Only one character format is automatically mapped to a DITA element:

```
[DITACHarTags]
Emphasis=i
```

See §24.4.4 [Mapping character formats to DITA inline elements](#) on page 465.

[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]
; 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 §24.4.5 [Assigning multiple typographic elements to a format](#) on page 467.

[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]
; format (para or char, wildcards OK) = level in DITA
; file required for the DITATag specified for this element.
Title=1
Heading*=3
```

See §24.5.12 [Specifying DITA element levels](#) on page 479.

[DITAParents] Only a few paragraph formats are automatically assigned DITA parents:

```
[DITAParents]
; format (para or char, wildcards OK) = required parents
Title=topic
Heading*=section
Numbered1=ol
Numbered=ol
Bulleted=ul
FigureTitle=fig
```

See §24.5.2 [Designating DITA ancestor elements](#) on page 472.

## 24.3 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](#)
- [CSS](#)
- [Context-sensitive help](#)

**Declaration** **DITA2Go** 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 §41.7.2 [Overriding settings in a DITA content model](#) on page 762 and §41.7.4 [Overriding declarations in a DITA map content model](#) on page 763.

**Standard XML options** The following XML settings cannot be overridden:

```
[HTMLOptions]
AlignAttributes = No
NoFonts = Yes

[Tables]
TableAttributes = 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
RemoveANames = Yes
```

```

ATagElement = xref
XMLEncoding = UTF-8
NumericCharRefs = No
FootInlineTag = fn
HardRetPara = No
RemoveEmptyParagraphs = Yes
RemoveEmptyTableParagraphs = Yes

```

See §22 [Converting to HTML/XHTML](#) on page 429.

*Style options* Keep empty paragraphs empty (if not removed):

```

[HTMLOptions]
EmptyParaContent =

```

See §30.2.5 [Providing content for empty paragraphs](#) on page 569.

CSS The CSS file **DITA2Go** 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. Setting [CSS]UseCSS=Yes also sets [DITAOptions]UseOutputClass=Yes; see §31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593.

**Note:** To include CSS class names as outputclass attributes, make sure your configuration file does *not* specify [CSS]WriteClassAttributes=No.

*Context-sensitive help*

By default, DITA output includes all context-sensitive help targets provided in your source document via **TopicAlias** PI 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:

§16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277

§24.10 [Including CSH targets in DITA XML](#) on page 491

## 24.4 Configuring DITA elements

*In this section:*

§24.4.1 [Understanding how DITA2Go delimits DITA elements](#) on page 460

§24.4.2 [Treating format names as DITA element names](#) on page 460

§24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460

§24.4.4 [Mapping character formats to DITA inline elements](#) on page 465

§24.4.5 [Assigning multiple typographic elements to a format](#) on page 467

§24.4.6 [Assigning attributes to DITA elements](#) on page 468

§24.4.7 [Preserving whitespace in block elements](#) on page 471

## 24.4.1 Understanding how DITA2Go delimits DITA elements

**DITA2Go** closes each element mapped from a paragraph format when a paragraph in that format ends. For example, even though a DITA list can be inside a `<p>` element, **DITA2Go** does not put it there; instead, the `<sl>` follows the `<p>`. Only elements that are marked as inline, including elements mapped from character formats, and inline images, are placed within a `<p>` element.

An interpolated block element stays open until **DITA2Go** encounters a paragraph that is not valid in that block.

Lists are identified by the format mapped to the list element it populates, such as `<li>` or `<sl>`; see §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460, or by the parent of the mapped element; see §24.5.2 [Designating DITA ancestor elements](#) on page 472. **DITA2Go** provides the wrappers around the list items and around the whole list.

## 24.4.2 Treating format names as DITA element names

If most formats are named for DITA elements, you can lessen the chore of mapping formats to elements by directing **DITA2Go** 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 §41.6 [Inspecting and correcting element types](#) on page 760.

However, 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 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 format name if valid in content
;   model).
UseFormatAsTag = Yes
```

When `UseFormatAsTag=Yes`, any 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:

§24.4.3.3 [Specifying a default element for unmapped paragraph formats](#) on page 462

§24.4.4.3 [Specifying a default element for unmapped character formats](#) on page 466.

## 24.4.3 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.

*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 §24.5 [Nesting DITA block elements](#) on page 471.

*In this section:*

§24.4.3.1 [Assigning DITA elements to paragraph formats](#) on page 461

- §24.4.3.2 [Omitting element tags for selected paragraph formats](#) on page 462
- §24.4.3.3 [Specifying a default element for unmapped paragraph formats](#) on page 462
- §24.4.3.4 [Omitting invalid tags for default DITA block elements](#) on page 462
- §24.4.3.5 [Overriding element mapping for paragraph formats](#) on page 463
- §24.4.3.6 [Providing aliases for paragraph formats](#) on page 463
- §24.4.3.7 [Mapping paragraph format aliases to different elements](#) on page 463
- §24.4.3.8 [Mapping paragraph format aliases algorithmically](#) on page 464
- §24.4.3.9 [Mapping several paragraphs formats to the same element](#) on page 465

### 24.4.3.1 Assigning DITA elements to paragraph formats

To map paragraph formats in your document to DITA elements, assign the element name to the format name:

```
[DITAParaTags]
; paragraph format (wildcards OK) = DITA element, can be
; overridden by a DITATag marker; or 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 paragraph format that is not mapped in [DITAParaTags] depends on the value of [DITAOptions]UseFormatAsTag; see §24.4.2 <a href="#">Treating format names as DITA element names</a> on page 460.  |
| <i>Do not map to footnote or table elements</i> | <b>DITA2Go</b> 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. If you are using the DITA Open Toolkit, see §24.9.4 <a href="#">Omitting &lt;xref&gt; elements from footnotes</a> on page 490. |
| <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 §24.5.5 <a href="#">Specifying alternate ancestries for the same element</a> on page 474.   |
| <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 §24.5.2 <a href="#">Designating DITA ancestor elements</a> on page 472. Definition lists can be derived from paragraph pairs.                                      |

*Add typographic elements* To add typographic elements (b, i, u, tt, sup, or sub) in addition to the element to which a format is mapped, see §24.4.5 [Assigning multiple typographic elements to a format](#) on page 467.

### 24.4.3.2 Omitting element tags for selected paragraph formats

To specify that a particular 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 §30.2.4 [Stripping paragraph properties](#) on page 568. 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.

**DITA2Go** 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, **DITA2Go** 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, **DITA2Go** does not try to interpolate. Instead, **DITA2Go** 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*. See §30.2.6 [Eliminating unwanted paragraphs](#) on page 569.

*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 **DITA2Go** will retain the original line breaks, instead of normalizing them as for HTML or XML. See §24.5.2 [Designating DITA ancestor elements](#) on page 472.

### 24.4.3.3 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 para formats that are
; not mapped in [DITAParaTags], default is "p".
DefParaElem = p
```

If your configuration file does not include a value for *DefParaElem*, **DITA2Go** uses one of the following as the element for an unmapped format: if *UseFormatAsTag=Yes* and the 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 §24.4.2 [Treating format names as DITA element names](#) on page 460.

### 24.4.3.4 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.



For enclosing block elements that allow mixed content, you can avoid this problem by directing **DITA2Go** 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:

§24.5.3 [Fixing up interpolated ancestries](#) on page 473

### 24.4.3.5 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 §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

Another alternative would be to bracket the text with **Config** PI markers, with content such as [ParaStyleCodeBefore]=<element> and [ParaStyleCodeAfter]=</element>; see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

**Note:** Mappings provided via [ParaStyleCode\*] settings or markers do not participate in any ancestry you specify for the element in question; see §24.5 [Nesting DITA block elements](#) on page 471.

### 24.4.3.6 Providing aliases for paragraph formats

To specify an alternate name, or alias, for a paragraph format:

```
[DITAAliases]
; paragraph format = format name to use in place of that
; paragraph format for DITA purposes, or DITA2Go 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, 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.

### 24.4.3.7 Mapping paragraph format aliases to different elements

Suppose your 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** PI marker in each instance, with content one of the other aliases: *Body2UList* or *Body2Quote*. Then you could specify the following in configuration file `d2dita.ini`:

```
[DITAParaTags]
Body2?list = p
must = lq
```

Instead of using a **DITAAlias** PI marker, you can provide differential mappings of the same format by assigning **DITA2Go** macros to the aliases; see §37 [Working with macros](#) on page 679.

### 24.4.3.8 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 format a **DITA2Go** 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 formats.

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 formats to DITA elements, allowing you to deal with cases that normal mapping cannot handle.

### 24.4.3.9 Mapping several paragraph formats to the same element

Suppose your 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
```

### 24.4.4 Mapping character formats to DITA inline 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:*

§24.4.4.1 [Assigning DITA elements to character formats](#) on page 465

§24.4.4.2 [Including typographic elements in addition to mapped formats](#) on page 466

§24.4.4.3 [Specifying a default element for unmapped character formats](#) on page 466

§24.4.4.4 [Overriding element mapping for character formats](#) on page 466

§24.4.4.5 [Using alternate character formats for menu cascades](#) on page 467

#### 24.4.4.1 Assigning DITA elements to character formats

To map character formats in your document to DITA inline elements, assign the element name to the format name:

```
[DITACHarTags]
; character format (wildcards OK) = DITA element, cannot be
; overridden by a DITATag marker; or format = No.
CharFmtName = elementname
```

For example:

```
[DITACHarTags]
Strong=b
Emphasis=i
BoldItalic=b
Subscript=sub
Superscript=sup
Mono=tt
Link=No
```

To specify that a particular character format should *not* be mapped to an element:

```
[DITACHarTags]
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`. **DITA2Go** automatically generates `<xref>` tags from the cross references in your document, based on the format, but you do not need to map the format

itself to any element. See §24.9 [Configuring cross references and links for DITA](#) on page 489.

The default element for a character format that is not mapped in [DITACharTags] is the element designated by DefCharElem; see §24.4.4.3 [Specifying a default element for unmapped character formats](#) on page 466. It is best to map each character format to the most specific element possible, which is not often the default element.

#### 24.4.4.2 Including typographic elements in addition to mapped formats

You can add typographic elements in addition to the element to which a 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 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 §24.4.6.6 [Providing outputclass attributes for all elements](#) on page 471. DITA asks for semantic, not presentational, tags. It is best to let CSS handle the presentation later.

See also:

§24.4.5 [Assigning multiple typographic elements to a format](#) on page 467

§30.7 [Managing typographic elements for HTML or XML](#) on page 579

#### 24.4.4.3 Specifying a default element for unmapped character formats

To specify a default element to use for unmapped character formats:

```
[DITAOptions]
; DefCharElem = element for char formats that are not
; mapped in [DITACharTags], default is "ph"
DefCharElem = ph
```

If your configuration file does not include a value for DefCharElem, **DITA2Go** uses one of the following as the element for an unmapped format: if UseFormatAsTag=Yes and the 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 §24.4.2 [Treating format names as DITA element names](#) on page 460.

#### 24.4.4.4 Overriding element mapping for character formats

If mapping a character format does not suffice for a phrase element, you can use **DITASTartElem** and **DITAEndElem** 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; **DITA2Go** provides the < > and </ >. You cannot use a **DITATag** marker to override the element-name mapping for an inline element.

#### 24.4.4.5 Using alternate character formats for menu cascades

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 §24.5.2 [Designating DITA ancestor elements](#) on page 472.

### 24.4.5 Assigning multiple typographic elements to a format

**DITA2Go** 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 `<b>` 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]
; character format (wildcards OK) = DITA typographic elements
CharFmtName = typelem1 typelem2 ...
```

Likewise for paragraph formats:

```
[DITAParaTypographics]
; 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 `[DITACCharTags]`. You *must* map the format to an element (not to `No`) in `[DITAParaTags]` or in `[DITACCharTags]`, then add the rest of the elements (but *not* the one already mapped) here. For example:

```
[DITACCharTags]
CodeBoldItal = tt
```

List the elements separated by spaces:

```
[DITACCharTypographics]
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><b><i>...</i></b></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 `<span class="codeboldital">`, which can be handled by CSS for selector `span.codeboldital`.

See §24.4.6.6 [Providing outputclass attributes for all elements](#) on page 471.

## 24.4.6 Assigning attributes to DITA elements

You can apply attributes to a DITA block or inline element by assigning *attribute="value"* pairs to the 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 §24.9 [Configuring cross references and links for DITA](#) on page 489.
- The `outputclass` attribute can be assigned to the root element only with a **DITATopicOutputclass** marker.

*In this section:*

§24.4.6.1 [Specifying a value for the id attribute](#) on page 468

§24.4.6.2 [Including an id attribute in every element](#) on page 469

§24.4.6.3 [Specifying attribute values for the root element of a topic](#) on page 469

§24.4.6.4 [Specifying attribute values for a block element or parent](#) on page 469

§24.4.6.5 [Specifying attribute values for an inline element](#) on page 470

§24.4.6.6 [Providing outputclass attributes for all elements](#) on page 471

*See also:*

§24.6 [Specifying options for tables in DITA XML](#) on page 480

### 24.4.6.1 Specifying a value for the id attribute

To specify an ID for a block element, place a **DITAElemID** PI marker in the paragraph. The content of the marker is the value of the `id` attribute. When you place a **DITAElemID** PI 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** PI marker instead; see §24.8.3 [Specifying the ID for a DITA topic](#) on page 487.

**Note:** The `id` attribute value must start with a letter.

**DITA2Go** provides a default `id` attribute for each of the following block elements:

```
<table> (all types)
<fig> (but not <image>)
<section>
<example>
<refsyn>
<fn>
<li>
```

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 PI marker or a **DITAElemID** PI marker in the target paragraph, and specify the link `type` attribute with a **DITALinkType** PI marker; see §24.9.5.3 [Specifying the <xref> type attribute](#) on page 491.

*Interpolated  
parent id attribute*

When the parent of the current block element is interpolated by **DITA2Go** (see §24.4.1 [Understanding how DITA2Go delimits DITA elements](#) on page 460), you cannot use a **DITAElemID** PI marker to specify an ID for that parent.

To specify an ID for the parent of the current block element, place a **DITAParentID** PI marker in the element, with content as follows:

```
parentname=parentid
```

Do not include spaces around the equals sign.

#### 24.4.6.2 Including an id attribute in every element

By default, **DITA2Go** automatically provides id attributes only for elements that require them; basically, any element that is a link target. To direct **DITA2Go** 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, **DITA2Go** 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** PI marker (see §24.4.6.1 [Specifying a value for the id attribute](#) on page 468).

#### 24.4.6.3 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 paragraph format for the topic title:

```
[DITATopicRootAttrs]
; 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 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** PI marker.

#### 24.4.6.4 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]
; para format (wildcards OK) = attributes
ParaFmt = attribute1="value1" attribute2="value2" ...
```

You can use **DITA2Go** 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** PI 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 **DITA2Go** macros for everything after the colon.

**Assign interpolated parent attributes**

To assign attributes to an interpolated parent of a block element:

```
[DITAParentAttributes]
; para format (wildcards OK) = parentname: attributes
ParaFmt = parentname: attribute1="value1" attribute2="value2" ...
```

You can use **DITA2Go** 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** PI 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** PI 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** PI markers for any of the following:

- The `id` attribute of the current element; use a **DITAElemID** PI marker instead. See §24.4.6.1 [Specifying a value for the id attribute](#) on page 468.
- The `id` attribute of an interpolated parent of the current element; use a **DITAParentID** PI marker instead. See §24.4.6.1 [Specifying a value for the id attribute](#) on page 468.
- Any `<xref>` element attributes; use **DITALink\*** PI markers instead. See §24.9.5 [Overriding <xref> attribute values](#) on page 490.

A **DITAAttribute** PI marker overrides settings in `[DITAParaAttributes]` and `[DITAParentAttributes]`, but does not override settings in `[DITACHarAttributes]` (see §24.4.6.5 [Specifying attribute values for an inline element](#) on page 470) or `[TableAttributes]` (see §24.6 [Specifying options for tables in DITA XML](#) on page 480).

### 24.4.6.5 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:

```
[DITCharAttributes]
; char format (wildcards OK) = attributes
CharFmt = attribute1="value1" attribute2="value2" ...
```

You cannot use markers to override settings in [DITCharAttributes].

#### 24.4.6.6 Providing outputclass attributes for all elements

By default, **DITA2Go** does not try to assign CSS classes to DITA elements (unless you set [CSS]UseCSS=Yes; see §31.4 [Specifying CSS file and link options](#) on page 593).

To direct **DITA2Go** 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, **DITA2Go** includes an outputclass attribute where allowed. The value of the attribute is the [ParaClasses] or [CharClasses] assignment (if any) for the format mapped to the element, otherwise the name of the format mapped to the element. See §31.7 [Assigning CSS classes](#) on page 599.

**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.

#### 24.4.7 Preserving whitespace in block elements

To make sure content will be processed with whitespace preserved:

```
[DITAPreformatted]
; element name = Yes (default for <codeblock>)
; or No (default for all other block elements)
codeblock = Yes
```

When *elementname*=Yes, **DITA2Go** processes the element in question with whitespace unchanged, as in HTML <pre> elements (see §30.9 [Configuring preformatted text for HTML/XML](#) on page 581).

## 24.5 Nesting DITA block elements

*In this section:*

- §24.5.1 [Understanding how DITA2Go determines element nesting](#) on page 472
- §24.5.2 [Designating DITA ancestor elements](#) on page 472
- §24.5.3 [Fixing up interpolated ancestries](#) on page 473
- §24.5.4 [Deciding when to fully specify ancestry](#) on page 474
- §24.5.5 [Specifying alternate ancestries for the same element](#) on page 474
- §24.5.6 [Avoiding invalid ancestries](#) on page 475
- §24.5.6 [Avoiding invalid ancestries](#) on page 475
- §24.5.8 [Configuring nested lists](#) on page 476
- §24.5.9 [Closing DITA ancestor elements](#) on page 476
- §24.5.10 [Opening DITA ancestor elements](#) on page 477

§24.5.11 [Configuring multi-paragraph list items](#) on page 478

§24.5.12 [Specifying DITA element levels](#) on page 479

See also:

§24.7.1 [Designating ancestors for <image> and <fig> elements](#) on page 482

## 24.5.1 Understanding how DITA2Go determines element nesting

For each element, **DITA2Go** considers whether that element can go inside the current parent element. If not, **DITA2Go** uses heuristic methods based on the possible parents, level limitations, and current context. To provide a parent element **DITA2Go** selects, in alphabetical order, the first wrapper element that can validly fit and that is permitted by your settings. *DITA2Go 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 §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460):

```
[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 **DITA2Go** 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** PI marker to explicitly set the level for an element; see §24.5.12 [Specifying DITA element levels](#) on page 479. However, for nested lists, use a different approach; see §24.5.5 [Specifying alternate ancestries for the same element](#) on page 474.

## 24.5.2 Designating DITA ancestor elements

For block elements such as <li> 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].

To specify required ancestors of elements mapped from 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 `<ol>` if and only if an `<ol>` is not already open; and starts a new `<li>` if and only if an `<li>` under the `<ol>` is not already open. To force a new `<ol><li>` for *Numbered1* paragraphs, you must also give the *Numbered1* paragraph format first-child status under both parent and grandparent elements; see §24.5.6 [Avoiding invalid ancestries](#) on page 475.

**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 `<li>` elements are needed under `<ul>` or `<ol>` 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** PI marker in the paragraph. Make the content of the marker the name(s) of the ancestor element(s), in hierarchical order. A **DITAParent** PI marker specifies the required ancestry for the current block element, overriding whatever is specified in `[DITAParents]`.

See also:

§24.7.1 [Designating ancestors for <image> and <fig> elements](#) on page 482

### 24.5.3 Fixing up interpolated ancestries

Creating DITA structure from 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, **DITA2Go** 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** PI marker in the paragraph for an isolated instance. In this example, the following mapping would produce better results:

```
[DITAParents]
Ref = refbody section
```

The **DITA2Go** search algorithm finds the shortest path, but that is not always the only shortest path, or the best path.

See also:

§24.4.3.4 [Omitting invalid tags for default DITA block elements](#) on page 462

§24.5.6 [Avoiding invalid ancestries](#) on page 475

## 24.5.4 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 **DITA2Go** 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 §24.5.5 [Specifying alternate ancestries for the same element](#) on page 474. 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 **DITA2Go** encounters a set of ancestors specified either in [DITAParents] or in a **DITAParent** marker, **DITA2Go** 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 <li>; in fact, in almost any block element. Unless you can make sure every block element that could precede a *Body* paragraph gets closed (see §24.5.9 [Closing DITA ancestor elements](#) on page 476), the *Body* <p> is likely to be nested in the preceding element.

## 24.5.5 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 **DITA2Go** 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** PI 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:*

§24.5.6 [Avoiding invalid ancestries](#) on page 475

## 24.5.6 Avoiding invalid ancestries

For the purpose of constructing ancestries, by default **DITA2Go** 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 **DITA2Go** 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 **DITA2Go** 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), a paragraph whose format is *Body* will be allowed as a `<body>` element in a reference topic, where `<body>` is not valid.

See also:

[§24.5.5 Specifying alternate ancestries for the same element](#) on page 474

## 24.5.7 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 [§24.5.2 Designating DITA ancestor elements](#) on page 472.

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 `<li>` element, which `<li>` element must be the first child of its `<ol>` 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 `<li>` 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 = dlentry
ParamTerm = plentry
```



To override the `[DITAFirst]` assignment for a given instance of a paragraph, place a **DITAFirst** PI marker in the paragraph. Make the content of the marker the name(s) of the desired ancestor element(s), in hierarchical order. A **DITAFirst** PI marker specifies that the current block element must be the first child of its listed ancestor elements, overriding whatever is specified in `[DITAFirst]`.

## 24.5.8 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 §24.5.5 [Specifying alternate ancestries for the same element](#) on page 474. 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 `<li><p>`, not just `<li>`, because the nested `<ul>` needs to be inside the `<li>`, 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 `<li>` 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
```

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.

## 24.5.9 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:*

§24.5.9.1 [Ending ancestor elements before the current block](#) on page 477

§24.5.9.2 [Ending ancestor elements after the current block](#) on page 477



### 24.5.9.1 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 `<li>`, 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]
; 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 §24.5.2 [Designating DITA ancestor elements](#) on page 472. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DITACloseBefore** PI 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** PI marker to override a `[DITACloseBefore]` setting when you want to close a higher (or lower) ancestor than the setting specifies.

### 24.5.9.2 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]
; 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 §24.5.2 [Designating DITA ancestor elements](#) on page 472. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DITACloseAfter** PI 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** PI marker to override a `[DITACloseAfter]` setting when you want to close a higher (or lower) ancestor than the setting specifies.

## 24.5.10 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:*

§24.5.10.1 [Starting ancestor elements before the current block](#) on page 477

§24.5.10.2 [Starting a new hierarchy after the current block](#) on page 478

### 24.5.10.1 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** PI 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** PI marker to override a [DITAOpenBefore] setting when you want to open a higher (or lower) ancestor than the setting specifies.

### 24.5.10.2 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** PI 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** PI marker to override a [DITAOpenAfter] setting when you want to open an element or hierarchy other than what the setting specifies.

## 24.5.11 Configuring multi-paragraph list items

By default, at the end of each paragraph **DITA2Go** closes the block element to which the paragraph format is mapped (see §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460). If any list items in your document include multiple paragraphs or sublists, you must make sure that each <li> 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 <li>.](#)

[Specify ancestry for each format.](#)

[Make each format first in <li>.](#)

[Make sure each list ends where it should.](#)

*Map formats to  
<p> instead of  
<li>*

Map all list-item paragraph formats to <p> rather than to <li>; 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 <li>*

Make sure each list-item paragraph is first in its <li> element:

```
[DITAFirst]
Numbered1 = ol li
Numbered = li
```

*Make sure each list ends where it should*

```
Bulleted = li
BulletedLast = li
```

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 `<li>`, 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** PI 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** PI marker in the following paragraph, with content `ul ol`.

## 24.5.12 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 paragraph formats that are mapped to the element (see §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460). However, for most nesting issues, you should use settings that specify ancestry rather than level; see §24.5.2 [Designating DITA ancestor elements](#) on page 472.

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 §24.5.8 [Configuring nested lists](#) on page 476.

To override the assigned level of a particular paragraph, place a **DITALevel** PI marker in the paragraph. A **DITALevel** PI 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** PI marker is a single integer.

*See also:*

§25.1.2 [Specifying topic levels in ditamaps](#) on page 495

## 24.6 Specifying options for tables in DITA XML

§24.6.1 [Designating ancestors for <table> elements](#) on page 480

§24.6.2 [Applying attributes to DITA tables](#) on page 480

§24.6.3 [Configuring DITA table components](#) on page 481

### 24.6.1 Designating ancestors for <table> elements

To specify the ancestor elements **DITA2Go** must use for <table> elements:

```
[DITAOptions]
; TableParents = parents for table tags, including simpleteable
; and others; default none (use content model), may include sets
; from [DITAElementSets].
TableParents =
```

List ancestors in hierarchical order; see §24.5.2 [Designating DITA ancestor elements](#) on page 472. You can include element sets, as well as single elements; see §24.5.5 [Specifying alternate ancestries for the same element](#) on page 474. If you do not specify any ancestor elements, **DITA2Go** 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 §33.2 [Defining sets of tables](#) on page 626). 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 PI marker, also; see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

### 24.6.2 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 paragraph or character format name or the [Class] setting for paragraph and character elements, just use [DITAOptions]UseOutputClass=Yes; see §24.4.6.6 [Providing outputclass attributes for all elements](#) on page 471.

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 §33.4.2 [Overriding attributes for selected tables](#) on page 633.

You can also specify an `outputclass` attribute for an individual table with a **TableOutputclass** PI marker in the text flow before the table anchor, or in the table title or first table row; see §33.1.1 [Understanding precedence of assignment methods](#) on page 625.

## 24.6.3 Configuring DITA table components

*In this section:*

§24.6.3.1 [Omitting ancestries of DITA table components](#) on page 481

§24.6.3.2 [Retaining empty paragraph tags in DITA table cells](#) on page 481

§24.6.3.3 [Specifying relative vs. absolute widths for table columns](#) on page 481

### 24.6.3.1 Omitting ancestries of DITA table components

Do not specify required parents (see §24.5.2 [Designating DITA ancestor elements](#) on page 472) for elements that are part of a table. **DITA2Go** 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>`.

### 24.6.3.2 Retaining empty paragraph tags in DITA table cells

By default, for DITA output **DITA2Go** 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 (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 §24.3 [Specifying general options for DITA](#) on page 458.

*See also:*

§33.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 640

### 24.6.3.3 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, **DITA2Go** 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 **DITA2Go** 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" = 81pt, and 1.5" = 108pt.

When TableColsRelative=No, **DITA2Go** produces instead:

```
<colspec colnum="1" colname="col1" colwidth="81pt" />
<colspec colnum="2" colname="col2" colwidth="108pt" />
```

## 24.7 Specifying options for images in DITA XML

*In this section:*

§24.7.1 [Designating ancestors for <image> and <fig> elements](#) on page 482

§24.7.2 [Specifying what to include in a <fig> wrapper](#) on page 483

§24.7.3 [Omitting size attributes from images for DITA output](#) on page 484

### 24.7.1 Designating ancestors for <image> and <fig> elements

To specify the ancestor elements **DITA2Go** 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 §24.5.2 [Designating DITA ancestor elements](#) on page 472. You can include element sets, as well as single elements; see §24.5.5 [Specifying alternate ancestries for the same element](#) on page 474. If you do not specify any ancestor elements, **DITA2Go** 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 §4.3 [Identifying files and elements](#) on page 76), or to the graphic group ID (see §32.4.1.4 [Creating named groups of graphics](#) on page 615). For example, to make sure icons in table cells have <entry> as a parent:

```
[GraphGroup]
ab01f853 = alerts
ab012c13 = alerts
ab00b5d3 = alerts

[DITAImageParents]
; image ID (may be group) = parents to be used for image/fig element.
alerts = entry
```

You can make a single `[DITAImageParents]` setting in an `HTMLConfig` marker, also; see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

*Sequence  
matters in  
element sets*

Although **DITA2Go** knows which elements are valid within other elements, **DITA2Go** has no idea at all about required sequences of elements. For example, if you set:

```
[DITAElementSets]
$iparents = conbody section entry example context choice
```

**DITA2Go** will always choose example over context when in <taskbody>. Where the image is valid in both <context> and <example>, **DITA2Go** lacks any real criterion for choosing one over the other. Instead, **DITA2Go** 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>.

## 24.7.2 Specifying what to include in a <fig> wrapper

When **DITA2Go** wraps image and title in a <fig> element, by default **DITA2Go** closes the <fig> element before moving on to the following content. To direct **DITA2Go** 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, **DITA2Go** 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
```

To specify that figure titles precede their images:

```
[DITAOptions]
; FigureTitleStartsFigure = No (default, title is below image),
; or Yes (title is above image)
FigureTitleStartsFigure = Yes
```

To prevent an image from being wrapped in a <fig> element, assign the NoFig format property to the enclosing 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 enclosing 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 enclosing 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 §3.6 [Using wildcards in configuration settings](#) on page 65.



### 24.7.3 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.

## 24.8 Configuring DITA topics

**DITA2Go** delimits DITA topics in your document according to paragraph formats you identify as <title> elements at DITA level 1.

*In this section:*

- §24.8.1 [Designating starting points for DITA topics](#) on page 484
- §24.8.2 [Specifying the DITA topic type](#) on page 486
- §24.8.3 [Specifying the ID for a DITA topic](#) on page 487
- §24.8.4 [Adjusting DITA topic IDs generated from file names](#) on page 488
- §24.8.5 [Specifying alternate titles for a DITA topic](#) on page 488
- §24.8.6 [Omitting a DITA topic from the TOC](#) on page 489

### 24.8.1 Designating starting points for DITA topics

**DITA2Go** bases starting points for DITA topics on the occurrence in your document of paragraph formats that have certain configuration settings. By default, **DITA2Go** also treats the very first paragraph format in each file as the start of a DITA topic.

*In this section:*

- §24.8.1.1 [Identifying starting elements for non-glossary topics](#) on page 484
- §24.8.1.2 [Identifying the starting element for glossary topics](#) on page 485
- §24.8.1.3 [Preventing the first paragraph format from starting a topic](#) on page 485

#### 24.8.1.1 Identifying starting elements for non-glossary topics

For topics of all built-in DITA types except `glossary`, the required starting element is <title>. **DITA2Go** 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 §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460.

2. Assign the format DITA level 1:

```
[DITALevels]
ParaFmt = 1
```

See §24.5.12 [Specifying DITA element levels](#) on page 479.

### 24.8.1.2 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 **DITA2Go** 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 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 §24.8.2.2 [Specifying a default DITA topic type](#) on page 486.

Setting the default topic type to `glossary` tells **DITA2Go** that the topics in the current file start with the paragraph format mapped to `glossterm` in `[DITAParaTags]`:

```
[DITAParaTags]
ParaFmt = glossterm
```

See §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460.

*Glossary mixed with other topics*

If the glossary for your document is in a 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
```

### 24.8.1.3 Preventing the first paragraph format from starting a topic

To prevent **DITA2Go** from forcing the first paragraph format in a 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`, **DITA2Go** 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`, **DITA2Go** 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 file is mapped to `title` and assigned level 1.

## 24.8.2 Specifying the DITA topic type

**DITA2Go** 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:*

- §24.8.2.1 [Understanding DITA topic type assignment precedence](#) on page 486
- §24.8.2.2 [Specifying a default DITA topic type](#) on page 486
- §24.8.2.3 [Specifying the DITA topic type with a paragraph format](#) on page 486
- §24.8.2.4 [Specifying the DITA topic type with a marker](#) on page 487

### 24.8.2.1 Understanding DITA topic type assignment precedence

The default type for every topic **DITA2Go** generates from your 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 24-1](#) shows the precedence of topic type assignment methods.

**Table 24-1** Precedence of DITA topic type assignment methods

Precedence	Method	Reference
Highest	<b>DITATopic</b> marker	<a href="#">24.8.2.4</a>
Intermediate	paragraph format	<a href="#">24.8.2.3</a>
Lowest	Default topic type	<a href="#">24.8.2.2</a>

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 §41 [Working with content models](#) on page 753.

### 24.8.2.2 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 paragraph format used in the topic.

To specify a different default topic type for a given chapter, include the `DefTopic` setting in a chapter configuration file named for the file; see §42.1.1 [Providing configuration files for individual ditamaps](#) on page 765.

### 24.8.2.3 Specifying the DITA topic type with a paragraph format

To assign a DITA topic type to a 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 §24.8.2.2 [Specifying a default DITA topic type](#) on page 486.
- 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 §24.8.1.2 [Identifying the starting element for glossary topics](#) on page 485). If necessary, modify your 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 **DITA2Go** encounters 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 **DITA2Go** encounters 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.

#### 24.8.2.4 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.

### 24.8.3 Specifying the ID for a DITA topic

To give a topic an ID, do one of the following:

- Place a **DITATopicID** PI marker in the topic; the content of the **DITATopicID** marker is the topic ID.
- Place a **FileName** PI marker in the topic; the marker content specifies both the topic ID and the base file name of the file that contains the topic (see §43.3.2 [Using PI markers to name output files](#) on page 782).

In the absence of either marker, the default topic ID is the base file name, adjusted as for CSS class names; see §24.8.4 [Adjusting DITA topic IDs generated from file names](#) on page 488.

If you do not insert markers for topic IDs, **DITA2Go** 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 file name (but only for the *first* topic in a file), include the following option in your project configuration file:

```
[DITAOptions]
; TopicID = id for topic, default is base file name
TopicID = someid
```

You can override the default ID with a **DITATopicID** PI marker, or with a **FileName** PI marker.

## 24.8.4 Adjusting DITA topic IDs generated from file names

By default, **DITA2Go** 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 **DITA2Go** 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/>

## 24.8.5 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](#)

[PI 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. **DITA2Go** 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 §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460.

*PI marker*

To use a PI marker for an alternate title, insert a **DITANavTitle** PI marker, a **DITASearchTitle** PI 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.

## 24.8.6 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 PI 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.

## 24.9 Configuring cross references and links for DITA

*In this section:*

§24.9.1 [Understanding how DITA2Go converts cross references](#) on page 489

§24.9.2 [Specifying an outputclass for cross-reference wrappers](#) on page 489

§24.9.3 [Linking to elements below topic level](#) on page 490

§24.9.4 [Omitting <xref> elements from footnotes](#) on page 490

§24.9.5 [Overriding <xref> attribute values](#) on page 490

### 24.9.1 Understanding how DITA2Go converts cross references

DITA allows cross references to <topic> (including each basic type), <section> (including <example> and <refsyn>), <table>, <fig>, <fn>, and <li>. No other elements. **DITA2Go** 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, **DITA2Go** automatically wraps the <xref> in a <ph> element, and assigns an outputclass attribute to the wrapper; see §24.9.2 [Specifying an outputclass for cross-reference wrappers](#) on page 489.

To provide a link destination for target elements that do not already contain a **DITAElemID** PI marker (see §24.4.6.1 [Specifying a value for the id attribute](#) on page 468), **DITA2Go** makes the ID of the target element the content of the first **HyperJump** PI marker in the element; or, in the absence of **HyperJump** PI markers, the numeric ID of the cross-reference marker.

### 24.9.2 Specifying an outputclass for cross-reference wrappers

When **DITA2Go** encounters a cross reference, index term, or footnote reference in a context where an <xref> tag would be invalid, the <xref> gets wrapped in a <ph> element. These **DITA2Go**-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.

### 24.9.3 Linking to elements below topic level

Normally, **DITA2Go** can use a **HyperJump** PI 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** PI marker in text just before the link, and a **DITAElemID** PI marker (or **DITAParentID** PI marker) in the target text.

The content of the **DITALinkElemID** PI marker must match the content of the **DITAElemID** PI marker or **DITAParentID** PI marker (see §24.4.6.1 [Specifying a value for the id attribute](#) on page 468). 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. **DITA2Go** places the **DITALinkElemID** PI marker content after the target topic ID in the `href` attribute of the `<xref>` element.

### 24.9.4 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
```

### 24.9.5 Overriding `<xref>` attribute values

You can insert PI markers in your 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:*

- §24.9.5.1 [Specifying the `<xref>` scope attribute](#) on page 490
- §24.9.5.2 [Specifying the `<xref>` format attribute](#) on page 491
- §24.9.5.3 [Specifying the `<xref>` type attribute](#) on page 491

#### 24.9.5.1 Specifying the `<xref>` scope attribute

By default, for most links **DITA2Go** 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 **HyperLink** PI marker, **DITA2Go** 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 **DITA2Go** 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, insert a **DITALinkScope** PI 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.

### 24.9.5.2 Specifying the `<xref>` format attribute

**DITA2Go** bases the value of the `<xref>` `format` attribute on the apparent destination of a link. To override this value, insert a **DITALinkFormat** PI 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** PI markers are needed only where **DITA2Go** built-in rules do not produce the correct value.

### 24.9.5.3 Specifying the `<xref>` type attribute

**DITA2Go** sets the `type` attribute of most generated `<xref>` elements to the DITA type (root element) of the destination topic. However, for links that use a **HyperLink** PI marker, **DITA2Go** omits the `type` attribute.

To specify the `type` attribute of the next cross reference or hypertext link, insert a **DITALinkType** PI 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:

§24.4.6.1 [Specifying a value for the `id` attribute](#) on page 468

## 24.10 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** PI markers. **DITA2Go** includes the content of **TopicAlias** PI 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`, **DITA2Go** processes the content of each **TopicAlias** PI 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 §16.10.2 [Specifying CSH mappings](#) on page 278.

## 24.11 Overriding DITA settings with markers

You might need to insert markers to override configuration settings for particular DITA topics and elements. **DITA2Go** provides numerous predefined marker types for this purpose, listed in [Table 24-2](#). Most of these marker types are intended to provide ways to cope with unusual situations.

**Table 24-2 Predefined marker types for DITA XML**

Marker type	Content	Reference
<b>DITACloseAfter</b>	Ancestor elements to be closed just after current block element ends	<a href="#">24.5.9.2</a>
<b>DITACloseBefore</b>	Ancestor elements to be closed just before current block element starts	<a href="#">24.5.9.1</a>
<b>DITACode</b>	XML code to be inserted at the marker location	
<b>DITAFirst</b>	Ancestor elements under which the current block element must be first	<a href="#">24.5.6</a>
<b>DITALevel</b>	Level where the current block element should appear in the DITA file	<a href="#">24.5.12</a>
<b>DITALinkElemID</b>	ID of the link target for the next <xref> element	<a href="#">24.9.3</a>
<b>DITALinkFormat</b>	Format attribute of the next <xref> element	<a href="#">24.9.5.2</a>
<b>DITALinkScope</b>	Scope attribute of the next <xref> element	<a href="#">24.9.5.1</a>
<b>DITALinkType</b>	Type attribute of the next <xref> element	<a href="#">24.9.5.3</a>
<b>DITANavTitle</b>	Alternate title for navigation use	<a href="#">24.8.5</a>
<b>DITANoToc</b>	Adds <code>toc="no"</code> to the <code>topicref</code> to the current topic, suppresses <code>navtitle</code>	<a href="#">24.8.6</a>
<b>DITAOpenAfter</b>	Elements to be opened just after current block element ends	<a href="#">24.5.10.2</a>
<b>DITAOpenBefore</b>	Enclosing elements to be opened just before current block element starts	<a href="#">24.5.10.1</a>
<b>DITAParent</b>	Required parents for the current block element	<a href="#">24.5.2</a>
<b>DITASearchTitle</b>	Alternate title for search-result use	<a href="#">24.8.5</a>
<b>DITATopic</b>	DTD type for the current topic	<a href="#">24.8.2</a>
<b>DITATopicID</b>	ID attribute for the current topic	<a href="#">24.8.3</a>

# 25 Configuring DITA maps

---

This section shows how to configure DITA maps. Topics include:

§25.1 [Configuring ditamaps](#) on page 493

§25.2 [Overriding map settings with PI markers](#) on page 498

*See also:*

§24 [Converting to DITA XML](#) on page 455

§41 [Working with content models](#) on page 753

## 25.1 Configuring ditamaps

*In this section:*

§25.1.1 [Specifying options for ditamaps](#) on page 493

§25.1.2 [Specifying topic levels in ditamaps](#) on page 495

§25.1.3 [Specifying roles for topics in ditamaps](#) on page 496

§25.1.4 [Adding relationship tables to ditamaps](#) on page 496

§25.1.5 [Providing navigation aids in ditamaps](#) on page 498

*See also:*

§41.7.4 [Overriding declarations in a DITA map content model](#) on page 763

### 25.1.1 Specifying options for ditamaps

*In this section:*

§25.1.1.1 [Choosing whether to overwrite ditamaps](#) on page 493

§25.1.1.2 [Choosing whether a ditamap references maps or topics](#) on page 494

§25.1.1.3 [Specifying the base file name for a ditamap](#) on page 494

§25.1.1.4 [Specifying a title for a chapter or book ditamap](#) on page 494

§25.1.1.5 [Specifying a navigation title for a ditamap](#) on page 495

#### 25.1.1.1 Choosing whether to overwrite ditamaps

By default, **DITA2Go** rewrites `.ditamap` files each time you run a DITA conversion. Unless the changes you make between conversion runs (changes either to your 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 **DITA2Go** 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

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, **DITA2Go** will not produce any DITA maps.

### 25.1.1.2 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 **DITA2Go** option is to have the book map reference topics directly rather than reference the chapter maps.

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.

### 25.1.1.3 Specifying the base file name for a ditamap

By default, the file name for each DITA map file is the base name of the 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 DITA file (see §42.1.1 [Providing configuration files for individual ditamaps](#) on page 765):

```
[DITAOptions]
; MapName = name to use (without extension) for the chapter ditamap.
MapName = othername
```

Alternatively, you can insert a **DITAMapName** PI marker in the DITA file. The content of the **DITAMapName** PI 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 PI marker to override the base name for a book map file.

### 25.1.1.4 Specifying a title for a chapter or book ditamap

*Arbitrary chapter  
map title*

To specify a title for a chapter map, include the following setting in a chapter-specific configuration file named for the DITA file (see §42.1.1 [Providing configuration files for individual ditamaps](#) on page 765):

```
[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`, **DITA2Go** 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** PI marker in the 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 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 PI marker to override the title for a book map file.

### 25.1.1.5 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 DITA file (see §42.1.1 [Providing configuration files for individual ditamaps](#) on page 765):

```
[DITAOptions]
; MapHead = navigation title for chapter map
MapHead = Title for use by navigation links
```

Alternatively, you can insert a **DITAMapHead** PI marker in the 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** PI marker, **DITA2Go** creates a <topichd> element at the start of the chapter map (effectively at map level 0) that contains the rest of the map entries.

## 25.1.2 Specifying topic levels in ditamaps

To specify map levels for topics, assign a level number to each format that you map to a <title> element at the topic level in [DITAParaTags]; see §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460. For example:

```
[DITAMapLevels]
; 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], **DITA2Go** uses any level assignments you have provided for those formats in [HelpContentsLevels]. See §16.4.3 [Setting contents levels for HTML-based Help](#) on page 251.

If you direct **DITA2Go** to nest topics, **DITA2Go** 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:

§24.5.12 [Specifying DITA element levels](#) on page 479

### 25.1.3 Specifying roles for topics in ditamaps

To specify the kind of entry (if any) each topic type should have in its chapter map:

```
[DITAMapUsage]
; 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 §24.4.3 [Mapping paragraph formats to DITA block elements](#) on page 460. These values have the following effects:

<code>Topic</code>	(Default) Include a <code>&lt;topicref&gt;</code> element in the map for this topic
<code>Head</code>	Include a <code>&lt;topichead&gt;</code> element only, with a title but no link
<code>None</code>	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.

### 25.1.4 Adding relationship tables to ditamaps

**DITA2Go** creates a relationship table for each chapter map generated from a DITA file.

In this section:

§25.1.4.1 [Understanding how DITA2Go creates relationship tables](#) on page 496

§25.1.4.2 [Excluding the ALink column from relationship tables](#) on page 496

§25.1.4.3 [Adding ALink rows to relationship tables](#) on page 497

§25.1.4.4 [Specifying one-way linking for a topic in a relationship table](#) on page 497

§25.1.4.5 [Specifying a collection-type attribute for each topic type](#) on page 497

#### 25.1.4.1 Understanding how DITA2Go creates relationship tables

By default, **DITA2Go** 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.

#### 25.1.4.2 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 **DITA2Go** to omit that column.

To prevent **DITA2Go** 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 document does not include ALink references, and you do not insert any **DITARelRow** PI marker (see §25.1.4.3 [Adding ALink rows to relationship tables](#) on page 497), **DITA2Go** does not produce a relationship table for the map.

#### 25.1.4.3 Adding ALink rows to relationship tables

To add a subject row to a relationship table, insert a **DITARelRow** PI marker in the topic. The content of a **DITARelRow** PI 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** PI markers are equivalent to **ALink** PI markers in Help systems.

#### 25.1.4.4 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** PI marker in the topic. The content of a **DITARelLinking** PI marker can be one of the following:

```
targetonly
sourceonly
```

#### 25.1.4.5 Specifying a collection-type attribute for each topic type

To specify a collection-type attribute for a topic type in a **DITA2Go**-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]`, **DITA2Go** 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 §25.1.4.3 [Adding ALink rows to relationship tables](#) on page 497).
- 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.

### 25.1.5 Providing navigation aids in ditamaps

To add navigation elements to the map entry for a topic, insert PI markers in the <title> paragraph of the topic. [Table 25-1](#) shows the content and placement of the navigation element for each PI marker type.

**Table 25-1 DITA map navigation elements from PI markers**

PI marker type	Marker content	Map element	Map placement
<b>DITAAncor</b>	id attribute	<anchor>	After the <topicref> of the topic
<b>DITANavref</b>	mapref attribute	<navref>	After the <topicref> of the topic
<b>DITAMapref</b>	href attribute	<topicref>	After the <topicref> of the topic
<b>DITALinkText</b>	text for the link	<linktext>	In the map <topicmeta>

The content of a **DITAMapref** PI marker should be a map file name. **DITA2Go** sets the format attribute of the resulting <topicref> element to "ditamap".

## 25.2 Overriding map settings with PI markers

You might need to insert PI markers to override configuration settings for particular DITA maps or for a bookmap. **DITA2Go** provides predefined PI marker types for this purpose, listed in [Table 25-2](#). Most of these PI marker types are intended to provide ways to cope with unusual situations.

**Table 25-2 Predefined PI marker types for DITA maps and bookmaps**

Marker type	Content	Ref.
<b>DITAAncor</b>	ID attribute of a map <anchor> element	<a href="#">25.1.5</a>
<b>DITALinkText</b>	Text of a map <linktext> element	<a href="#">25.1.5</a>
<b>DITAMapHead</b>	Navigation title for the current chapter map	<a href="#">25.1.1.5</a>
<b>DITAMapName</b>	Base name (without extension) of map file for current chapter	<a href="#">25.1.1.3</a>
<b>DITAMapref</b>	href attribute of a map <topicref> element	<a href="#">25.1.5</a>
<b>DITAMapTitle</b>	Text of <title> element inserted at map level 0	<a href="#">25.1.1.4</a>
<b>DITANavref</b>	mapref attribute of a map <navref> element	<a href="#">25.1.5</a>
<b>DITARelLinking</b>	Linking attribute for a topic in a relationship table	<a href="#">25.1.4.4</a>
<b>DITARelRow</b>	ALink subject name for a row in a relationship table	<a href="#">25.1.4.3</a>

# 26 Converting to DocBook XML

---

**DITA2Go** generates DocBook XML output. This section shows how to configure DocBook-specific options. Topics include:

- §26.1 [Generating DocBook XML with DITA2Go](#) on page 499
- §26.2 [Setting up a DocBook XML project](#) on page 500
- §26.3 [Specifying general options for DocBook](#) on page 502
- §26.4 [Configuring DocBook elements](#) on page 504
- §26.5 [Nesting DocBook block elements](#) on page 511
- §26.6 [Designating ancestors for table elements](#) on page 519
- §26.7 [Specifying options for figure elements](#) on page 520
- §26.8 [Overriding DocBook settings with PI markers](#) on page 521

*See also:*

- §41 [Working with content models](#) on page 753

## 26.1 Generating DocBook XML with DITA2Go

Before you set up a **DITA2Go** 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 **DITA2Go** to play in producing DocBook output.

*In this section:*

- §26.1.1 [Understanding what you need to know about DocBook](#) on page 499
- §26.1.2 [Understanding what information you must supply](#) on page 499

### 26.1.1 Understanding what you need to know about DocBook

To use **DITA2Go** 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 **DITA2Go User's Guide**. You have to know *what* you want; then perhaps we can tell you *how* to make it happen with **DITA2Go**.

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 **DITA2Go** 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>

### 26.1.2 Understanding what information you must supply

**DITA2Go** does not try to validate the output; you must use a validating parser to check output validity. However, **DITA2Go** 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 DITA document.

**DITA2Go** support for DocBook requires you to supply the following kinds of information in addition to your DITA document:

### DTD properties

### DITA mappings

*DTD properties* **DITA2Go** 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 §41.2.1 [Obtaining a copy of a built-in content-model](#) on page 754. However, the only valid purpose for modifying a built-in content model would be to correct settings for element types. See §41.6 [Inspecting and correcting element types](#) on page 760.

To replace a content model, use free command-line utility **dttd2ini** to generate a content model from another DTD, and produce a content-model configuration file for your DocBook project. See §41.2.2 [Generating a content model from a DTD](#) on page 754.

*DITA mappings* You must map formats to DocBook elements. This information goes into configuration file `d2docbook.ini`, and possibly into chapter-specific configuration files. You might have to use PI marker in your DITA document to provide information such as topic IDs, element names, and attributes, in cases where these items cannot be derived from the document.

## 26.2 Setting up a DocBook XML project

To add or change any of the options described in this section, edit configuration file `_d2docbook.ini`, located in the project directory. Or, to apply the changes to all of your DocBook XML projects, edit the configuration template referenced by `_d2docbook.ini`:

```
%omsyshome%\d2g\local\config\local_d2docbook_config.ini.
```

See §39.4 [Deciding which configuration file to edit](#) on page 734.

*In this section:*

§26.2.1 [Creating a DocBook project](#) on page 500

§26.2.2 [Specifying DocBook output options](#) on page 501

### 26.2.1 Creating a DocBook project

*Easier:* use the **DITA2Go** Project Manager to start a new project; see §2.1 [Creating a DITA2Go conversion project](#) on page 39.

To create a DocBook XML project without using the **DITA2Go** Project Manager:

1. Create a directory for DocBook output, separate from the directory where your DITA document is located.
2. Copy configuration file `d2docbook.ini` from your **DITA2Go** `config` directory (see §1.3.1 [Set up a framework for Omni Systems applications](#) on page 29), or from an existing **DITA2Go** project, to your newly created output directory:

**DITA2Go** copies a new project configuration file, `_d2docbook.ini`, to your project directory. 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 `_d2docbook.ini`; see §3.1 [Working with DITA2Go configuration files](#) on page 49.

## 26.2.2 Specifying DocBook output options

When you set up a DocBook project, **DITA2Go** includes settings for several output options. You can specify values for these options and a few more in configuration file `_d2docbook.ini`.

*In this section:*

§26.2.2.1 [Changing the DocBook output file extension](#) on page 501

§26.2.2.2 [Specifying content model and root element](#) on page 501

§26.2.2.3 [Specifying book file options](#) on page 501

### 26.2.2.1 Changing the DocBook output file extension

To change the file extension for DocBook chapter files:

```
[Setup]
FileSuffix = .ext
```

The default extension is `.ent`.

### 26.2.2.2 Specifying content model and root element

By default, **DITA2Go** 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 §41 [Working with content models](#) on page 753.

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 §41.2.2 [Generating a content model from a DTD](#) on page 754.

### 26.2.2.3 Specifying book file options

By default, **DITA2Go** writes a DocBook book file with the DocBook chapter files included as entity references.

To omit the book file:

```
[DocBookOptions]
; WriteBookFile = Yes (default, write one file) or No
; Sets UseDOCTYPE and UseXMLRoot to No for the chapter entity files.
WriteBookFile = No
```

See §26.3.1 [Configuring entity information for DocBook XML](#) on page 502.

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
; extension .xml
BookFileName = YourBookFileName.xml
; BookFileTitle = text of title for book, default is literally
```

```
; Your Book Title
BookFileTitle = Your Book Title
```

## 26.3 Specifying general options for DocBook

This section lists DocBook-specific default values and recommended options for XHTML and XML configuration settings.

*In this section:*

§26.3.1 [Configuring entity information for DocBook XML](#) on page 502

§26.3.2 [Configuring links for DocBook XML](#) on page 502

§26.3.3 [Configuring tables for DocBook XML](#) on page 503

§26.3.5 [Configuring footnotes for DocBook XML](#) on page 503

### 26.3.1 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, **DITA2Go** sets UseDOCTYPE and UseXMLRoot to No for chapter entity files; see §26.2.2 [Specifying DocBook output options](#) on page 501.

### 26.3.2 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, 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:*

§23.4 [Configuring links for generic XML](#) on page 454

### 26.3.3 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
```

### 26.3.4 Retaining empty paragraph tags in DocBook table cells

By default, for DocBook output **DITA2Go** 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 (contains only empty paragraphs) would become just `<entry></entry>` in DocBook XML output.

*See also:*

§33.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 640

### 26.3.5 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:*

§30.10 [Converting footnotes to HTML or XML](#) on page 581

## 26.4 Configuring DocBook elements

*In this section:*

- §26.4.1 [Treating format names as element names](#) on page 504
- §26.4.2 [Mapping paragraph formats to DocBook elements](#) on page 504
- §26.4.3 [Mapping character formats to DocBook elements](#) on page 507
- §26.4.4 [Assigning ID attributes to DocBook block elements](#) on page 508
- §26.4.5 [Assigning attributes other than ID to DocBook elements](#) on page 510

### 26.4.1 Treating format names as element names

To map 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 format name if valid in content
; model).
UseFormatAsTag = Yes
```

When UseFormatAsTag=Yes, any 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:

- §26.4.2.2 [Specifying a default element for unmapped paragraph formats](#) on page 505
- §26.4.3.2 [Specifying a default element for unmapped character formats](#) on page 508.

### 26.4.2 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:*

- §26.4.2.1 [Assigning DocBook elements to paragraph formats](#) on page 504
- §26.4.2.2 [Specifying a default element for unmapped paragraph formats](#) on page 505
- §26.4.2.3 [Omitting invalid tags for default DocBook block elements](#) on page 505
- §26.4.2.4 [Overriding element mapping for paragraph formats](#) on page 506
- §26.4.2.5 [Providing aliases for paragraph formats](#) on page 506

#### 26.4.2.1 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]
; paragraph format (wildcards OK) = DocBook element, can be
; overridden by a DocBookTag marker; or format = No.
ParaFmtName = elementname
```

*Default element*

The default element for a 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 §26.4.2.2 [Specifying a default element for unmapped paragraph formats](#) on page 505.



*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 §26.5.2 [Designating DocBook ancestor elements](#) on page 512. Definition lists can be derived from paragraph pairs, possibly with run-in headings for the term.

*Omit element  
mapping*

To specify that a particular 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 **DITA2Go** will retain the original line breaks, instead of normalizing them as for HTML or XML.

See §26.5.2 [Designating DocBook ancestor elements](#) on page 512.

### 26.4.2.2 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 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`, **DITA2Go** uses one of the following as the element for an unmapped format: if `UseFormatAsTag=Yes` and the 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 §26.4.1 [Treating format names as element names](#) on page 504.

### 26.4.2.3 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 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 **DITA2Go** 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:

§26.5.3 [Fixing up interpolated ancestries](#) on page 513

#### 26.4.2.4 Overriding element mapping for paragraph formats

To override the element-name mapping for a given paragraph, insert a **DocBookTag** PI 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 §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

Another alternative would be to bracket the text with **Config** PI markers, with content such as [ParaStyleCodeBefore]=<element> and [ParaStyleCodeAfter]=</element>; see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

**Note:** Mappings provided via [ParaStyleCode\*] settings or markers do not participate in any ancestry you specify for the element in question; see §26.5 [Nesting DocBook block elements](#) on page 511.

#### 26.4.2.5 Providing aliases for paragraph formats

To specify an alternate name, or alias, for a paragraph format:

```
[DocBookAlias]
; paragraph format = 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** PI 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** PI 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.

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 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** PI 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 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
```

### 26.4.3 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:*

- §26.4.3.1 [Assigning DocBook elements to character formats](#) on page 507
- §26.4.3.2 [Specifying a default element for unmapped character formats](#) on page 508
- §26.4.3.3 [Overriding element mapping for character formats](#) on page 508

#### 26.4.3.1 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]
; character format (wildcards OK) = DocBook element, cannot be
; overridden by a DocBookTag marker; or format = No.
CharFmtName = elementname
```

To specify that a particular 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`. **DITA2Go** automatically generates `<xref>` tags from the cross

references in DITA, based on the format, but you do not need to map the format itself to any element.

The default element for a character format that is not mapped in [DocBookCharTags] is the element designated by DefCharElem; see §26.4.3.2 [Specifying a default element for unmapped character formats](#) on page 508. It is best to map each character format to the most specific element possible, which is not often the default element.

### 26.4.3.2 Specifying a default element for unmapped character formats

To specify a default element to use for unmapped character formats:

```
[DocBookOptions]
; DefCharElem = element for 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, **DITA2Go** uses one of the following as the element for an unmapped format: if UseFormatAsTag=Yes and the 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 §26.4.1 [Treating format names as element names](#) on page 504.

### 26.4.3.3 Overriding element mapping for character formats

If mapping a character format does not suffice for an inline element, you can use **DocBookStartElem** and **DocBookEndElem** PI 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; **DITA2Go** provides the < > and </ >. You cannot use a **DocBookTag** PI marker to override the element-name mapping for an inline element.

## 26.4.4 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 **DITA2Go** automatically assign an ID to each block element derived from a paragraph (or to an interpolated parent of such a block element). You can also use PI markers to assign IDs to specific block elements, or to override a **DITA2Go**-assigned ID.

*In this section:*

- §26.4.4.1 [Understanding how DITA2Go creates ID attribute values](#) on page 508
- §26.4.4.2 [Providing IDs for block elements](#) on page 509
- §26.4.4.3 [Providing IDs for interpolated parents of block elements](#) on page 509
- §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510

### 26.4.4.1 Understanding how DITA2Go creates ID attribute values

When **DITA2Go** assigns an ID to a block element in DocBook, the value of the ID attribute is a combination of the **DITA2Go** FileID of the file being processed and the ObjectID of the paragraph from which the element was generated.

If a particular block element requires an ID value, **DITA2Go** looks for a **DocBookElemID** PI marker in the paragraph from which the element was generated. If that paragraph does not contain a **DocBookElemID** PI marker, **DITA2Go** uses the content of the first **HyperAnchor** PI marker in the paragraph as the ID for the element. If there is no

**HyperAnchor** PI marker, and you have requested automatic ID assignment for that element, **DITA2Go** assigns an ID.

You can override an automatically assigned ID for a particular block element by inserting either a **HyperAnchor** PI marker or a **DocBookElemID** PI marker in the paragraph from which the element is generated; see §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.

#### 26.4.4.2 Providing IDs for block elements

To direct **DITA2Go** to automatically include an ID attribute in each instance of a block element mapped from a particular paragraph format:

```
[DocBookParaIDs]
; para format = No (default, no ID set automatically) or Yes
ParaFmt = Yes
```

When *ParaFmt*=Yes, **DITA2Go** 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*), **DITA2Go** 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 **HyperAnchor** PI marker is already present in the paragraph from which the element was generated, you must insert a **DocBookElemID** PI marker to provide an ID for the element; see §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.

**Note:** If you list the same paragraph format in [DocBookParentIDs], **DITA2Go** changes the value in [DocBookParaIDs] to No; see §26.4.4.3 [Providing IDs for interpolated parents of block elements](#) on page 509.

You can override the value of an automatically assigned ID attribute with a **HyperAnchor** PI marker or a **DocBookElemID** PI marker inserted in the paragraph; see §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.

#### 26.4.4.3 Providing IDs for interpolated parents of block elements

To direct **DITA2Go** to include an ID attribute in each instance of an interpolated parent of an element mapped from a particular paragraph format:

```
[DocBookParentIDs]
; para format = single parent element for which the ID
; of the para should be used.
ParaFmt = ParentElement
```

When you assign a parent element in [DocBookParentIDs], **DITA2Go** 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 **DITA2Go** 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 §26.4.4.2 [Providing IDs for block elements](#) on page 509. And it also eliminates any ID assigned to the child via **DocBookElemID** PI marker; see §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.

You can override the value of an automatically assigned parent ID attribute with a **DocBookParentID** PI marker inserted in the child paragraph; see §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.

#### 26.4.4.4 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** PI marker in the paragraph. The content of the marker is the value of the `id` attribute. You can also override any **DITA2Go**-assigned ID with a **DocBookElemID** PI marker.

To specify an ID for the interpolated parent of the block element derived from a particular paragraph, place a **DocBookParentID** PI marker in the paragraph, with content as follows:

```
parentname=parentid
```

Do not include spaces around the equals sign.

See also:

§26.4.4.2 [Providing IDs for block elements](#) on page 509

### 26.4.5 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 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 PI markers; attributes of elements mapped from character formats cannot be overridden with PI markers.

In this section:

§26.4.5.1 [Specifying attribute values for a block element or ancestor](#) on page 510

§26.4.5.2 [Specifying attribute values for an inline element](#) on page 511

#### 26.4.5.1 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 PI 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]
; para format (wildcards OK) = attributes
ParaFmt = attribute1="value1" attribute2="value2" ...
```

You can use **DITA2Go** 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** PI 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 **DITA2Go** macros for everything after the colon.

*Assign  
interpolated  
parent attributes*

To assign attributes to an interpolated parent of a block element:

```
[DocBookParentAttributes]
; para format (wildcards OK) = parentname: attributes
ParaFmt = parentname: attribute1="value1" attribute2="value2" ...
```

You can use **DITA2Go** 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** PI 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 PI  
markers*

Use **DocBookAttribute** PI 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** PI markers for either of the following:

- The *id* attribute of the current element; use a **DocBookElemID** PI marker instead. See §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.
- The *id* attribute of an interpolated parent of the current element; use a **DocBookParentID** PI marker instead. See §26.4.4.4 [Specifying an ID for an individual block element or parent](#) on page 510.

A **DocBookAttribute** PI marker overrides settings in [DocBookParaAttributes] and [DocBookParentAttributes], but does not override settings in [DocBookCharAttributes] (see §26.4.5.2 [Specifying attribute values for an inline element](#) on page 511).

### 26.4.5.2 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]
; char format (wildcards OK) = attributes
CharFmt = attribute1="value1" attribute2="value2" ...
```

You cannot use markers to override settings in [DocBookCharAttributes].

## 26.5 Nesting DocBook block elements

*In this section:*

- §26.5.1 [Understanding how DITA2Go determines element nesting](#) on page 512
- §26.5.2 [Designating DocBook ancestor elements](#) on page 512
- §26.5.3 [Fixing up interpolated ancestries](#) on page 513
- §26.5.4 [Deciding when to fully specify ancestry](#) on page 513
- §26.5.5 [Specifying alternate ancestries for the same element](#) on page 514
- §26.5.6 [Specifying first-child status for nested elements](#) on page 514
- §26.5.7 [Specifying full ancestry for nested sections](#) on page 515
- §26.5.8 [Closing DocBook ancestor elements](#) on page 516



[§26.5.9 Opening DocBook ancestor elements](#) on page 517

[§26.5.10 Configuring multi-paragraph list items](#) on page 517

[§26.5.11 Specifying DocBook element levels](#) on page 518

See also:

[§26.6 Designating ancestors for table elements](#) on page 519

[§26.7 Specifying options for figure elements](#) on page 520

## 26.5.1 Understanding how DITA2Go determines element nesting

For each element, **DITA2Go** considers whether that element can go inside the current parent element. If not, **DITA2Go** 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 [§26.4.2 Mapping paragraph formats to DocBook elements](#) on page 504):

```
[DocBookParaTags]
Step* = step
```

As soon as **DITA2Go** encounters a paragraph format that is not valid in `<procedure>`, the parent tag is closed.

For problem cases, you can use a **DocBookLevel** PI marker to explicitly set the level for an element; see [§26.5.11 Specifying DocBook element levels](#) on page 518. However, for nested lists, use a different approach; see [§26.5.5 Specifying alternate ancestries for the same element](#) on page 514.

Leaving *any* paragraph or character format unmapped to a parent is risky; **DITA2Go** might interpolate the name of a DocBook element that does not do what you want.

## 26.5.2 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]
; 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 §26.5.6 [Specifying first-child status for nested elements](#) on page 514.

**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** PI 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]`.

### 26.5.3 Fixing up interpolated ancestries

Creating DocBook structure from 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, **DITA2Go** 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** PI marker in the paragraph for an isolated instance. In this example, the following mapping would produce better results:

```
[DocBookParents]
Ref = refentry refsect1
```

The **DITA2Go** search algorithm finds the shortest path, but that is not always the only shortest path, or the best path.

See also:

§26.4.2.3 [Omitting invalid tags for default DocBook block elements](#) on page 505

### 26.5.4 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 **DITA2Go** 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 §26.5.7 [Specifying full ancestry for nested sections](#) on page 515.

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 §26.5.5 [Specifying alternate ancestries for the same element](#) on page 514.

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 **DITA2Go** encounters a set of ancestors specified either in [DocBookParents] or in a **DocBookParent** PI marker, **DITA2Go** 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 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 §26.5.8 [Closing DocBook ancestor elements](#) on page 516), the *Body* <para> is likely to be nested in the preceding element.

## 26.5.5 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 **DITA2Go** 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** PI 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 PI marker.

## 26.5.6 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]
; 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 §26.5.2 [Designating DocBook ancestor elements](#) on page 512.

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** PI 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].

## 26.5.7 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:

§26.5.5 [Specifying alternate ancestries for the same element](#) on page 514

§26.5.6 [Specifying first-child status for nested elements](#) on page 514

## 26.5.8 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:*

§26.5.8.1 [Ending ancestor elements before the current block](#) on page 516

§26.5.8.2 [Ending ancestor elements after the current block](#) on page 516

### 26.5.8.1 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]
; 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 §26.5.2 [Designating DocBook ancestor elements](#) on page 512. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DocBookCloseBefore** PI 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** PI marker to override a `[DocBookCloseBefore]` setting when you want to close a higher (or lower) ancestor than the setting specifies.

### 26.5.8.2 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]
; 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 §26.5.2 [Designating DocBook ancestor elements](#) on page 512. You can list as many possible ancestors as necessary; order is not important.

For individual cases, you can insert a **DocBookCloseAfter** PI 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** PI marker to override a `[DocBookCloseAfter]` setting when you want to close a higher (or lower) ancestor than the setting specifies.

## 26.5.9 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:*

§26.5.9.1 [Starting ancestor elements before the current block](#) on page 517

§26.5.9.2 [Starting a new hierarchy after the current block](#) on page 517

### 26.5.9.1 Starting ancestor elements before the current block

To open interpolated ancestor elements before starting the current block:

```
[DocBookOpenBefore]
; 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** PI 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** PI marker to override a [DocBookOpenBefore] setting when you want to open a higher (or lower) ancestor than the setting specifies.

### 26.5.9.2 Starting a new hierarchy after the current block

To open a new element or hierarchy of elements after the current block ends:

```
[DocBookOpenAfter]
; 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** PI 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** PI marker to override a [DocBookOpenAfter] setting when you want to open an element or hierarchy other than what the setting specifies.

## 26.5.10 Configuring multi-paragraph list items

By default, at the end of each paragraph **DITA2Go** closes the block element to which the paragraph format is mapped (see §26.4.2 [Mapping paragraph formats to DocBook elements](#) on page 504). 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
```



	<pre> Numbered = para Bulleted = para BulletedLast = para </pre>
<i>Specify ancestry for each format</i>	<p>Designate the appropriate ancestors for each type of list element:</p> <pre> [DocBookParents] Numbered1 = orderedlist listitem Numbered = orderedlist listitem Bulleted = itemizedlist listitem BulletedLast = itemizedlist listitem </pre>
<i>Make each format first in &lt;listitem&gt;</i>	<p>Make sure each list-item paragraph is first in its &lt;listitem&gt; element:</p> <pre> [DocBookFirst] Numbered1 = orderedlist listitem Numbered = listitem Bulleted = listitem BulletedLast = listitem </pre>
<i>Make sure each list ends where it should</i>	<p>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 &lt;listitem&gt;, that paragraph will be included in the list item. To prevent including the following paragraph, you can explicitly close the list:</p> <pre> [DocBookCloseAfter] BulletedLast = itemizedlist listitem </pre> <p>Or insert a <b>DocBookCloseAfter</b> PI marker in the last list-item paragraph, with content <code>itemizedlist orderedlist</code>.</p> <p>As an alternative, you can make sure the list closes before the following paragraph:</p> <pre> [DocBookCloseBefore] Body = itemizedlist orderedlist </pre> <p>Or insert a <b>DocBookCloseBefore</b> PI marker in the following paragraph, with content:</p> <pre> itemizedlist orderedlist </pre>

## 26.5.11 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 paragraph formats that are mapped to the element (see §26.4.2 [Mapping paragraph formats to DocBook elements](#) on page 504). However, for most nesting issues, you should use settings that specify ancestry rather than level; see §26.5.2 [Designating DocBook ancestor elements](#) on page 512. 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]
; para format (wildcards OK) = level in DocBook 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** PI marker in the paragraph. A **DocBookLevel** PI 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** PI marker is a single integer.

## 26.6 Designating ancestors for table elements

To specify the ancestor elements **DITA2Go** 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 §26.5.2 [Designating DocBook ancestor elements](#) on page 512. You can include element sets, as well as single elements; see §26.5.5 [Specifying alternate ancestries for the same element](#) on page 514. If you do not specify any ancestor elements, **DITA2Go** 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 §33.2 [Defining sets of tables](#) on page 626). 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` PI marker, also; see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

## 26.7 Specifying options for figure elements

*In this section:*

§26.7.1 [Deciding what to include in a figure element](#) on page 520

§26.7.2 [Specifying ancestry for figure elements](#) on page 520

§26.7.3 [Omitting size attributes from images for DocBook](#) on page 521

### 26.7.1 Deciding what to include in a figure element

When **DITA2Go** wraps image and title in a `<figure>` element, by default **DITA2Go** closes the `<figure>` element before moving on to the following content. To direct **DITA2Go** 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, **DITA2Go** 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
```

To specify that figure titles precede their images:

```
[DocBookOptions]
; FigureTitleStartsFigure = No (default, title is below image),
; or Yes (title is above image)
FigureTitleStartsFigure = Yes
```

### 26.7.2 Specifying ancestry for figure elements

To specify the ancestor elements **DITA2Go** 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 §26.5.2 [Designating DocBook ancestor elements](#) on page 512. You can include element sets, as well as single elements; see §26.5.5 [Specifying alternate ancestries for the same element](#) on page 514. If you do not specify any ancestor elements, **DITA2Go** 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 §4.3 [Identifying files and elements](#) on page 76), or to the graphic group ID (see §32.4.1.4

Creating named groups of graphics on page 615). 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 §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

*Sequence matters  
in element sets*

Although **DITA2Go** knows which elements are valid within other elements, **DITA2Go** has no idea at all about required sequences of elements. For example, if you set:

```
[DocBookElementSets]
$iparents = section entry example graphic
```

**DITA2Go** will always choose `example` over `graphic`. Where the image is valid in both `<graphic>` and `<example>`, **DITA2Go** lacks any real criterion for choosing one over the other. Instead, **DITA2Go** 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>`.

### 26.7.3 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.

## 26.8 Overriding DocBook settings with PI markers

You might need to insert PI markers in your document to override configuration settings for particular DocBook elements. **DITA2Go** provides predefined PI marker types for this purpose, listed in [Table 26-1](#). Most of these PI marker types are intended to provide ways to cope with unusual situations.

**Table 26-1** Predefined PI marker types for DocBook

PI marker type	Content	Ref.
<b>DocBookAlias</b>	Alternate name for format for the current block	<a href="#">26.4.2.5</a>
<b>DocBookAttribute</b>	Attributes other than ID of a non- <code>&lt;xref&gt;</code> block element or parent	<a href="#">26.4.5.1</a>
<b>DocBookCloseAfter</b>	Ancestor elements to be closed just after current block element ends	<a href="#">26.5.8.2</a>
<b>DocBookCloseBefore</b>	Ancestor elements to be closed just before current block element starts	<a href="#">26.5.8.1</a>

**Table 26-1 Predefined PI marker types for DocBook (continued)**

PI marker type	Content	Ref.
<b>DocBookCode</b>	XML code to be inserted at the marker location	
<b>DocBookElemID</b>	ID attribute for the current block element	<a href="#">26.4.4.4</a>
<b>DocBookEndElem</b>	Tag name for an inline element, to place at the end of the span	<a href="#">26.4.3.3</a>
<b>DocBookFirst</b>	Ancestor elements under which the current block element must be first	<a href="#">26.5.6</a>
<b>DocBookLevel</b>	Level where the current block element should appear in the DocBook file	<a href="#">26.5.11</a>
<b>DocBookOpenAfter</b>	Elements to be opened just after current block element ends	<a href="#">26.5.9.2</a>
<b>DocBookOpenBefore</b>	Enclosing elements to be opened just before current block element starts	<a href="#">26.5.9.1</a>
<b>DocBookParent</b>	Required ancestors for the current block element	<a href="#">26.5.2</a>
<b>DocBookParentID</b>	ID of the interpolated parent of the current block element	<a href="#">26.4.4.4</a>
<b>DocBookStartElem</b>	Tag name for an inline element, to place at the start of the span	<a href="#">26.4.3.3</a>
<b>DocBookTag</b>	Element name mapping for the current block (not inline) element	<a href="#">26.4.2.4</a>

# 27 Splitting and extracting files

---

This section shows how **DITA2Go** can divide a DITA file or map, based on criteria you provide, to convert to HTML; and how **DITA2Go** can extract sections out of the middle of a file, to create excerpts in their own files. Topics include:

- §27.1 [Splitting and extracting vs. chunking](#) on page 523
- §27.2 [Chunking DITA maps](#) on page 523
- §27.3 [Splitting files](#) on page 526
- §27.4 [Extracting files](#) on page 528
- §27.5 [Identifying split and extract files](#) on page 530
- §27.6 [Inserting HTML code in split and extract files](#) on page 534
- §27.7 [Referencing split and extract files](#) on page 536
- §27.8 [Customizing and replacing extracts](#) on page 537

## 27.1 Splitting and extracting vs. chunking

Basically, you have two ways to chop up the entire set of DITA map topics into pages for HTML: use DITA chunking, or use the **DITA2Go** split mechanism.

DITA *chunking* uses the same methods as **DITA2Go** file splitting. However, in effect, chunking inserts **Split** PI markers for you, and sets file names per any `copy-to` attributes; see §27.2 [Chunking DITA maps](#) on page 523. You can use splitting and extracting in addition to chunking.

When you *split* a DITA file, each piece of the file becomes a file in its own right, typically addressing a single topic; see §27.3 [Splitting files](#) on page 526.

When you *extract* part of a DITA 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; see §27.4 [Extracting files](#) on page 528.

## 27.2 Chunking DITA maps

Chunking takes place in DITA map files, not in topic files. Unless you specify chunking either in map files or in configuration settings, an entire map results in a single output file.

*In this section:*

- §27.2.1 [Choosing between splitting and chunking](#) on page 523
- §27.2.2 [Specifying a chunking policy](#) on page 524
- §27.2.3 [Providing a page break between title and TOC](#) on page 525
- §27.2.4 [Producing a single HTML file that includes generated lists](#) on page 525

### 27.2.1 Choosing between splitting and chunking

By default, **DITA2Go** splits documents at DITA topic boundaries to produce an output file for every topic. To use DITA chunking to assort topics into output files instead:

```
[Chunking]
; SplitTopicFiles = Yes (default, produce an output file for every
; topic, override any chunkby attributes), or No (use DITA chunking
```

```
; if any, otherwise produce one output file)
SplitTopicFiles = No
```

When `SplitTopicFiles=Yes`, **DITA2Go** overrides chunking attributes and produces a separate output file for each topic. `SplitTopicFiles=Yes` has the same effect as setting `ChunkBy=Topic`; see §27.2.2 [Specifying a chunking policy](#) on page 524.

When `SplitTopicFiles=No`, **DITA2Go** uses DITA chunking, if any is specified. In the absence of chunking, **DITA2Go** produces one large file. At that point, you can use the **DITA2Go** split mechanism instead; see §27.3 [Splitting files](#) on page 526.

To disable chunking entirely:

```
[Chunking]
; DisableChunking = No (default, process chunking attribute normally),
; or Yes (ignore chunking attribute)
DisableChunking = Yes
```

When `DisableChunking=Yes`, **DITA2Go** ignores the chunking attribute, and instead goes by the setting for `SplitTopicFiles`, or by the `Split` settings in `[HTMLParaStyles]`; see §27.3 [Splitting files](#) on page 526.

**Note:** For generated files (see §14.1 [Understanding how DITA2Go produces lists](#) on page 197) you do not have to disable chunking in order for `SplitTopicFiles` to be effective.

## 27.2.2 Specifying a chunking policy

These settings establish a default chunking policy, to determine how topics are assorted into output files for each referenced document. The settings take effect for the entire map, unless chunking attributes are already present on the map element, and except where chunking attributes are present on individual `topicref` elements.

**Note:** There is no default policy if you do *not* set a chunking policy. However, in the absence of any settings to the contrary, **DITA2Go** uses chunk by topic for HTML output and chunk by document for RTF output.

Default output file names are overridden by `copy-to` attributes.

You can specify default policies for:

[Aggregation of topics](#)

[Selection of topics](#).

*Aggregation of  
topics*

To specify chunking by topic or by document:

```
[Chunking]
; ChunkBy = Document (put all topics in one output file,
; or Topic (put each topic into its own file).
ChunkBy = Topic
```

**DITA2Go** distributes topics according to the value of `ChunkBy`:

Document	All referenced topics go into one output file, named for the top-level map or document.
Topic	Each referenced topic goes into its own output file, named for the topic ID.

There is no default value for `ChunkBy`.

*Selection of  
topics*

To specify selection of all topics, of just the referenced topic, or of the referenced topic and all its descendants:

```
[Chunking]
; ChunkSel = Document (include all topics in the document,
; Topic (include only the topic id'd in the <topicref>, or
; Branch (include the id'd (or first) topic and all of its children).
ChunkSel = Topic
```

**DITA2Go** includes topics according to the value of `ChunkSel`:

<code>Document</code>	All topics in the referenced document are included in a single file.
<code>Topic</code>	A separate file is produced for each referenced topic.
<code>Branch</code>	The file for each referenced topic includes all topics logically nested in (descended from) that topic.

`ChunkSel` has no meaning for files that contain just one topic, only for those that contain nested topics, or topics wrapped in `<dita>`.

There is no default value for `ChunkSel`.

### 27.2.3 Providing a page break between title and TOC

If your HTML output includes material that precedes the TOC (a cover page, splash screen, or title), by default **DITA2Go** suppresses the page break between that material and the TOC. This is because if you use the default values for both of the following settings:

```
[Chunking]
SplitTopicFiles = Yes
DisableChunking = No
```

you might be left with output that provides no way to navigate from the preceding material to the TOC.

To keep the page break between the preceding material and the TOC:

```
[Contents]
; KeepTOCWithTitlePage = Yes (default, suppress page break between
; title page and TOC) or No (keep page break)
KeepTOCWithTitlePage = No
```

Specify `KeepTOCWithTitlePage=No` if the title content already includes a link to the TOC.

### 27.2.4 Producing a single HTML file that includes generated lists

To produce a monolithic HTML file that includes the TOC and any other generated lists (LOF, LOT, generated index), turn off both chunking and splitting for HTML output:

```
[Chunking]
SplitTopicFiles = No
DisableChunking = Yes
```

When `SplitTopicFiles=No` and `DisableChunking=Yes`, **DITA2Go** ignores any chunking set on the maps themselves, and does not split any files. **DITA2Go** also suppresses the automatic split between the TOC and the first topic.

If you have also specified a generated index, **DITA2Go** puts the CSS for the index in the single `<head>`, puts the JavaScript for it (if any) at the top of the `<body>`, and adds the index content at the end of the 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 §27.6.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 535.

## 27.3 Splitting files

*In this section:*

§27.3.1 [Designating split points](#) on page 526

§27.3.2 [Managing split points](#) on page 527

### 27.3.1 Designating split points

You can designate places to split a DITA file based on the occurrence of any of the following:

[Paragraph or character formats in text](#) on page 526

[HeadingPara = Split Title Paragraph formats in tables](#) on page 526

[DITA PI markers](#) on page 526

[Hypertext alert markers](#) on page 527

Although you can use all these methods in the same DITA 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.)

*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 element mapped to that format appears in your DITA 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  
Paragraph  
formats in tables*

Normally **DITA2Go** 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 element, not at the paragraph in the table.

*DITA PI markers*

You can add DITA PI marker type **Split** to your DITA document, and insert a **Split** marker wherever you want a split point. However, if you insert a **Split** PI marker in a paragraph whose format is also assigned the `Split` property, the **Split** PI marker is ignored.

A **Split** PI marker must be placed to the left of any **Title**, **FileName**, or **Cross-Ref** marker that occurs in the same paragraph.

See §38 [Working with processing instructions](#) on page 717 for more information.

*Hypertext alert markers* You can mark a split point in your document by inserting a **HyperAlert** PI marker that contains only the word “split”.

## 27.3.2 Managing split points

You can selectively disable splitting to keep content together, such as at the start of a DITA file, or when a single heading would be the only content.

*In this section:*

§27.3.2.1 [Preventing splits that leave dangling headings](#) on page 527

§27.3.2.2 [Keeping headings together when other content intervenes](#) on page 528

### 27.3.2.1 Preventing splits that leave dangling headings

Suppose you have designated the following criterion for split points:

```
[HTMLParaStyles]
Heading2 = Split
```

Now suppose your DITA 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, to avoid a title-only topic, 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]
; 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
```

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 §27.3.2.2 [Keeping headings together when other content intervenes](#) on page 528).

**Note:** It is not a good idea to use SmartSplit in conjunction with trails of links; see §29.2.1 [Understanding trails of links](#) on page 555.

### 27.3.2.2 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.

## 27.4 Extracting files

You can direct **DITA2Go** 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. You can use this feature to move material such as a large figure or table into its own HTML page.

*In this section:*

§27.4.1 [Enabling and disabling extract processing](#) on page 528

§27.4.2 [Delimiting material to extract](#) on page 529

### 27.4.1 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].

## 27.4.2 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 PI markers. For either of these methods to take effect, you must also enable extract processing; see §27.4.1 [Enabling and disabling extract processing](#) on page 528.

*In this section:*

§27.4.2.1 [Using existing paragraph formats to delimit extracts](#) on page 529

§27.4.2.2 [Creating special paragraph formats to delimit extracts](#) on page 529

§27.4.2.3 [Using markers to delimit extracts](#) on page 530

*See also:*

§27.4.1 [Enabling and disabling extract processing](#) on page 528

### 27.4.2.1 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 §27.4.1 [Enabling and disabling extract processing](#) on page 528.

For examples, see §27.8.4 [Specifying extracts: an example](#) on page 543.

### 27.4.2.2 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. Insert an element with `outputclass=ExEnd` just after material to be extracted.

4. Make the *ExStart* and *ExEnd* paragraphs conditional for on-line use, so they do not appear in your regular print files.
5. 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 **DITA2Go**. 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 §27.4.1 [Enabling and disabling extract processing](#) on page 528.

### 27.4.2.3 Using markers to delimit extracts

You can specify the boundaries of an extract with carefully placed PI markers. **DITA2Go** ignores the content of these markers, so you can use the content for comments:

<b>ExtrStart</b>	First part of an extract; insert in the first paragraph in the extract, to the left of any <b>Title</b> , <b>FileName</b> , or <b>Cross-Ref</b> marker that occurs in the same paragraph.
<b>ExtrFinish</b>	Last part of an extract; insert in the last paragraph of the extract.
<b>ExtrEnd</b>	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** PI 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 §27.4.1 [Enabling and disabling extract processing](#) on page 528.

## 27.5 Identifying split and extract files

You can specify titles and meta information for split and extract files. **DITA2Go** 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:*

- §27.5.1 [Understanding how split and extract files are named](#) on page 530
- §27.5.2 [Specifying page titles for split or extract files](#) on page 531
- §27.5.3 [Supplying <meta> text for split or extract files](#) on page 534

### 27.5.1 Understanding how split and extract files are named

When **DITA2Go** splits a DITA file, the result is a series of HTML files. If you direct **DITA2Go** to include a map-based TOC (see §6.12 [Specifying options for maps](#) on page 105), that TOC becomes the first output file. Otherwise, the first topic becomes the first output file. If you specify a name for the first file, either via TOC naming or via chunking, **DITA2Go** uses the name you specify; in that case an empty file with the name

of the original map is left behind and can be ignored. *Do not rename this file*, because it is required for subsequent conversions. **DITA2Go** names the rest of the split files, and all extracted files, as follows:

- If there is a `copy-to` attribute on the `topicref` in the map, the attribute value becomes the name of the file.
- If there is no `copy-to` attribute:
  - If chunking is specified, **DITA2Go** derives a name from the topic ID of the affected topic.
  - If chunking is not specified, the file gets a name that begins with `xx` and ends with a three- or four-digit internally assigned ID number.

## 27.5.2 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 DITA PI 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:*

[§27.5.2.1 Understanding title assignment precedence](#) on page 531

[§27.5.2.2 Assigning a title with a paragraph format](#) on page 532

[§27.5.2.3 Specifying a title prefix or suffix](#) on page 532

[§27.5.2.4 Assigning a title with a file name](#) on page 533

[§27.5.2.5 Assigning a title with a marker](#) on page 533

[§27.5.2.6 Assigning a default title](#) on page 533

[§27.5.2.7 Assigning a computed title](#) on page 534

### 27.5.2.1 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 27-1](#) shows which method takes precedence if more than one method applies to a given output file.

**Table 27-1 Precedence of HTML page titles**

Precedence	Method	Comments
Highest	<code>[Titles]</code>	Titles assigned in this section cannot be overridden
Intermediate	<code>[HTMLParaStyles]</code> or <b>Title</b> , <b>Config</b> , or <b>HTMConfig</b> marker	If you use both methods for a given split or extract part, whichever occurs first in the source file (the <b>Title</b> , <b>Config</b> , or <b>HTMConfig</b> marker or an instance of the <code>Title</code> paragraph format) takes precedence
Lowest	<code>[HTMLOptions]</code>	Any other method overrides an <code>[HTMLOptions]Title</code> value

*Default title* If you do not specify any page titles, the default title for each output file is *Test File from DITA2Go*; any other title specification overrides this default. If some of your output files

show *Test File from DITA2Go* as the title, this means you did not manage to specify titles for those files. See §27.5.2.6 [Assigning a default title](#) on page 533.

### 27.5.2.2 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 §27.5.2.3 [Specifying a title prefix or suffix](#) on page 532.

*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 **DITA2Go** 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 §30.2.4 [Stripping paragraph properties](#) on page 568.

See also:

§27.8.2 [Customizing title text for extracts](#) on page 538

§30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566

### 27.5.2.3 Specifying a title prefix or suffix

If you assign titles with paragraph formats (see §27.5.2.2 [Assigning a title with a paragraph format](#) on page 532), 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.



- 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, **DITA2Go** removes exactly one of them (see §3.4 [Understanding the rules for configuration settings](#) on page 62). If you want a single leading space for the title suffix, supply exactly two spaces after the equals sign.

### 27.5.2.4 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 **DITA2Go** (see §27.5.1 [Understanding how split and extract files are named](#) on page 530), 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 DITA file from which the HTML file was generated:

```
[Titles]
*=<$$_basefile>
```

This setting would provide the base name (without extension) of the DITA 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 §27.5.2.7 [Assigning a computed title](#) on page 534.

*See also:*

- §3.6 [Using wildcards in configuration settings](#) on page 65
- §27.3.4 [Using predefined macro variables](#) on page 691

### 27.5.2.5 Assigning a title with a marker

You can insert a DITA PI marker of type **Title** in the first paragraph of a split or extract, and supply the text of the title as the marker content. 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 §27.3.1 [Designating split points](#) on page 526), or custom marker type **ExtrStart**; and specify the title as the content. See §38 [Working with processing instructions](#) on page 717 for more information.

### 27.5.2.6 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 DITA2Go*.

### 27.5.2.7 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 DITA 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:*

§37.1 [Defining and invoking macros](#) on page 679

§37.3.3 [Incrementing and decrementing macro variables](#) on page 690

§37.6.3 [Displaying expression results in output](#) on page 702

## 27.5.3 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 §22.4.6 [Supplying content for the <meta> tag](#) on page 436 for more information.

## 27.6 Inserting HTML code in split and extract files

Split and extract files require the usual HTML <head> and <body> sections. **DITA2Go** provides code for these sections, just as for any other HTML output file. You can specify additional HTML code, including macros, for **DITA2Go** to insert at several points in these HTML sections.

*In this section:*

§27.6.1 [Choosing how to insert code in extracts](#) on page 534

§27.6.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 535

§27.6.3 [Using special sections to insert code in extracts](#) on page 536

### 27.6.1 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 27-2](#). 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 27-2 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	<a href="#">27.8.1</a>
Intermediate	Configuration section	Assign code to the paragraph format designated to start the extract (see <a href="#">27.4.2.1</a> )	<a href="#">27.6.3</a>
Lowest	Configuration keyword	Assign code to an [Inserts] keyword	<a href="#">27.6.2</a>

The methods in [Table 27-2](#) are listed in order of precedence. For example, if you use both an **ExtrHead** PI marker containing HTML code and supply an [Inserts]ExtrHead entry specifying HTML code, **DITA2Go** 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.

## 27.6.2 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 27-3](#) 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 27-3 Basic macro-insertion keywords and locations**

[Inserts] keyword	Location of macro in HTML file
TopicBreak	Before the <title> element (between topics) when files are not split
Head	Within the <head> element, after the <title> element
HeadEnd	At the very end of the <head> element
Frames	Between <head> and <body> elements, for framesets (split files only)
Top	At the beginning of the <body> element.
Bottom	Just before the end of the <body> element.
End	After the end of the <body> element (to close noframes; split files only)

*[Inserts] file-type  
prefixes*

[Table 27-4](#) 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 27-4 Keyword prefixes for split or extract code insertion**

[Inserts] 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 Top or Bottom
Extr	Extract file; use only with Head, Top, or Bottom

*[Inserts]* location/file-type variants Table 27-5 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 27-5** 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).

### 27.6.3 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 PI 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 §27.6.1 [Choosing how to insert code in extracts](#) on page 534.

For examples, see §27.8.4 [Specifying extracts: an example](#) on page 543.

## 27.7 Referencing split and extract files

You can use the predefined macro variables listed in [Table 27-6](#) 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.

**Table 27-6 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 <code>[Graphics]ExtrGraphSuffix</code> ; use for thumbnails (see §27.8.3 <a href="#">Replacing extracts with links in the parent file</a> on page 539)
	<code>\$\$_extrgraphid</code>	<b>DITA2Go</b> 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 §27.8.3 <a href="#">Replacing extracts with links in the parent file</a> on page 539.
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.

## 27.8 Customizing and replacing extracts

*In this section:*

§27.8.1 [Using PI markers for extract processing](#) on page 537

§27.8.2 [Customizing title text for extracts](#) on page 538

§27.8.3 [Replacing extracts with links in the parent file](#) on page 539

§27.8.4 [Specifying extracts: an example](#) on page 543

### 27.8.1 Using PI markers for extract processing

You can use predefined PI marker types to supply most extract-file properties. [Table 27-7](#) lists the predefined marker types for extracts.

**Table 27-7 Predefined PI marker types for extracts**

Marker type	Purpose	Content use	Placement	Ref.
<b>ExtrBottom</b>	Insert HTML code	Last item in extract <code>&lt;body&gt;</code>	Anywhere in the extract*	<a href="#">27.6.1</a>
<b>ExtrDisable</b>	Turn off extract processing	Ignored	After all or a group of extracts	<a href="#">27.4.1</a>
<b>ExtrEnable</b>	Turn on extract processing	Ignored	Before all or a group of extracts	<a href="#">27.4.1</a>

**Table 27-7 Predefined PI marker types for extracts (continued)**

Marker type	Purpose	Content use	Placement	Ref.
<b>ExtrEnd</b>	End an extract;	Ignored	After end of last paragraph in extract, and before end of following paragraph	<a href="#">27.4.2</a>
<b>ExtrFinish</b>	Mark last part of extract	Ignored	In the last paragraph of the extract	<a href="#">27.4.2</a>
<b>ExtrHead</b>	Insert HTML code	Placed in extract <head>	Anywhere in the extract*	<a href="#">27.6.1</a>
<b>ExtrReplace</b>	Insert HTML code	Replaces extract in parent file	Anywhere in the extract*	<a href="#">27.8.3</a>
<b>ExtrStart</b>	Begins extract	Ignored	In the first paragraph of the extract; can be used in [Extr*] sections	<a href="#">27.4.2</a> , <a href="#">27.6.3</a>
<b>ExtrTop</b>	Insert HTML code	First item in extract <body>	Anywhere in the extract*	<a href="#">27.6.1</a>

\* To avoid maintenance headaches, pick a consistent location, such as at the start of the first paragraph.

**DITA2Go** 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 §27.6.1 [Choosing how to insert code in extracts](#) on page 534.

## 27.8.2 Customizing title text for extracts

In the [ExtrTitle] section you can assign text for the extract title to the ExtrStart format or **ExtrStart** PI 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 §27.7 [Referencing split and extract files](#) on page 536.

For example, suppose your DITA document contains an element mapped to 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].

## 27.8.3 Replacing extracts with links in the parent file

*In this section:*

§27.8.3.1 [Assigning replacement code](#) on page 539

§27.8.3.2 [Using thumbnails for links to illustrations in HTML](#) on page 540

§27.8.3.3 [Supplying properties for extracted graphics](#) on page 542

### 27.8.3.1 Assigning replacement code

In the [ExtrReplace] section you can assign HTML code, including macros, to the ExtrStart format or **ExtrStart** PI 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** PI 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 27-8](#) 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 §27.8.3.2.3 [Providing scaled thumbnails](#) on page 541.

**Table 27-8 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	<a href="#">27.8.3.2.2</a>
<\$\$_extrgraphclass>	CSS class name to use in <\$_extrthumb> macro	<a href="#">27.8.3.2.3</a>
<\$\$_extrgraphhigh>	Thumbnail height in pixels, for use in <\$_extrthumb> macro	<a href="#">27.8.3.2.3</a>
<\$\$_extrgraphtarget>	target attribute for window used by <\$_extrthumb>	<a href="#">27.8.3.2.3</a>
<\$\$_extrgraphid>	<b>DITA2Go</b> internal name of the first graphic in an extract; use to reference properties	<a href="#">27.8.3.3</a>
<\$\$_extrgraphwide>	Thumbnail width in pixels, for use in <\$_extrthumb> macro	<a href="#">27.8.3.2.3</a>
<\$\$_extrfile>	Extract file name	<a href="#">27.8.3.3</a>
<\$\$_extrtitle>	Extract title	<a href="#">27.8.3.3</a>

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 §27.8.3.2 [Using thumbnails for links to illustrations in HTML](#) on page 540.

### 27.8.3.2 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:*

§27.8.3.2.1 [Choosing a thumbnail method](#) on page 540

§27.8.3.2.2 [Providing separate thumbnails](#) on page 540

§27.8.3.2.3 [Providing scaled thumbnails](#) on page 541

§27.8.3.2.4 [Including text with a thumbnail](#) on page 542

#### 27.8.3.2.1 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.

#### 27.8.3.2.2 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
```

<i>Name thumbnail after output graphic</i>	Except for the suffix, each thumbnail must have the same name as the corresponding output graphic.
<i>Place thumbnail in project directory</i>	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: <pre>D:\MyDoc\graphics\jaguar.jpg</pre> You must provide the following thumbnail for <code>jaguar.jpg</code> : <pre>D:\MyDoc\graphics\jaguartn.jpg</pre>
<i>Thumbnail not found</i>	If <b>DITA2Go</b> 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.
<i>Graphic not present</i>	If there is no graphic in the extract, the value of <code>&lt;\$_extrgraph&gt;</code> for that extract is NULL, and the literal name <code>extrgraph</code> appears in the replacement code wherever you specified <code>&lt;\$_extrgraph&gt;</code> .

### 27.8.3.2.3 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`, **DITA2Go** 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 **DITA2Go** internal namespaces.

<i>Preserve aspect ratio</i>	If you want to use a reduced size for thumbnails, but not all images have the same aspect ratio, set only one of <code>ExtrGraphHigh</code> or <code>ExtrGraphWide</code> to the number of pixels you want, and set the other to 0 (zero).
<i>Predefined macro &lt;\$_extrthumb&gt;</i>	For convenience you can use built-in macro <code>&lt;\$_extrthumb&gt;</code> , which is defined as follows: <pre>&lt;p class="&lt;\$_extrgraphclass&gt;"&gt;&lt;a href="&lt;\$_extrfile&gt;"   target="&lt;\$_extrgraphtarget&gt;"&gt;&lt;img src="&lt;\$_extrgraph&gt;" \   &lt;\$_if (\$\$_extrgraphhigh &gt; 0)&gt; height="&lt;\$_extrgraphhigh&gt;"&lt;\$_endif&gt;\</pre>

```
<$_if ($$_extrgraphwide > 0)> width="<$$_extrgraphwide">"<$_endif>\n
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>
```

#### 27.8.3.2.4 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 §27.8.3.2.3 [Providing scaled thumbnails](#) on page 541), 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 **DITA2Go** generate CSS each time. You could also give the table a class, and define its properties in the CSS file.

#### 27.8.3.3 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** PI 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 §32.8.2 [Adjusting image size for selected graphics](#) on page 621):

```
[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 §37.4 [Using multiple-value list variables](#) on page 695) 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>
```

## 27.8.4 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">
```

# 28 Creating HTML links

---

This section shows how to provide basic links in HTML output. Topics include:

- §28.1 [Understanding sources of links](#) on page 545
- §28.2 [Specifying link appearance](#) on page 545
- §28.3 [Specifying link destination](#) on page 549
- §28.4 [Creating jumps to particular windows for HTML](#) on page 550
- §28.5 [Converting DITA cross-reference links to HTML](#) on page 551
- §28.6 [Linking to other files and other DITA2Go projects](#) on page 553
- §28.7 [Linking to external destinations](#) on page 554

See also:

- §29 [Providing navigation in HTML](#) on page 555

## 28.1 Understanding sources of links

DITA2Go creates HTML links from the following items in DITA:

<i>Cross references:</i>	Cross references are converted to links to cross-reference sources.
<i>Hypertext internal links:</i>	Hypertext links are activated in HTML.
<i>Hypertext external links:</i>	Links to external files are activated in HTML; see §28.7 <a href="#">Linking to external destinations</a> on page 554.
<i>Paragraph formats:</i>	Links connecting a hierarchy of headings can form a “breadcrumb trail”; see §29.2 <a href="#">Generating trails of links</a> on page 555.

## 28.2 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:

- §28.2.1 [Specifying link colors](#) on page 545
- §28.2.2 [Specifying link class](#) on page 546
- §28.2.3 [Assigning link attributes with PI markers](#) on page 547
- §28.2.4 [Specifying link properties with macros](#) on page 548
- §28.2.5 [Replacing problem characters in links](#) on page 548
- §28.2.6 [Forcing link text to lowercase](#) on page 549

### 28.2.1 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 §31 [Setting up CSS for HTML](#) on page 591). 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="#FFFFE1" 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 §28.2.2 [Specifying link class](#) on page 546.

## 28.2.2 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 PI marker in DITA XML, or via paragraph-format assignment in the **DITA2Go** configuration file. To change just one or two links, probably a PI marker is easier. To change many links, you might want to use a special `@outputclass` for the paragraph format for the text where the links occur.

*In this section:*

§28.2.2.1 [Assigning a link class with a PI marker](#) on page 546

§28.2.2.2 [Assigning a link class with a paragraph format](#) on page 547

### 28.2.2.1 Assigning a link class with a PI marker

You can use DITA PI marker **LinkClass** to assign a CSS class to a single link, as follows:

1. For each link:
  - 1.1. Place a **LinkClass** PI marker in your document, just before the link.
  - 1.2. 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** PI marker with content:

```
traffic
```

CSS class `traffic` would be applied (only) to the next link after the PI marker:

```
<a class="traffic" ...>link text</a>
```



See also:

§28.2.3 [Assigning link attributes with PI markers](#) on page 547

§34.3.3 [Assigning WAI link attribute values with PI markers](#) on page 652

§38.3 [Adding attributes with PI markers](#) on page 721

### 28.2.2.2 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 §31.5 [Understanding how CSS affects other options](#) on page 596.

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>
```

### 28.2.3 Assigning link attributes with PI markers

To assign an HTML attribute to a link, just before the link insert a PI 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. **DITA2Go** puts the attribute and its value in the generated `<a>` tag.

See also:

§28.2.2.1 [Assigning a link class with a PI marker](#) on page 546

§34.3.3 [Assigning WAI link attribute values with PI markers](#) on page 652

§38.3 [Adding attributes with PI markers](#) on page 721

## 28.2.4 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. **DITA2Go** 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 §37 [Working with macros](#) on page 679.

## 28.2.5 Replacing problem characters in links

Some characters that are acceptable in DITA hypertext links and cross references cause problems for browsers; for example, HTML insists on all-lowercase IDs. **DITA2Go** processes DITA hypertext links and cross references to ensure acceptable IDs, similar to the way CSS class names are processed; see §31.7.1 [Understanding CSS class name restrictions](#) on page 600.

*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, **DITA2Go** 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: **DITA2Go** removes all spaces without replacing them.

## 28.2.6 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 **DITA2Go** 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 §43.3.3.3 [Creating special paragraph formats to name output files](#) on page 784).

## 28.3 Specifying link destination

*In this section:*

- §28.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 549
- §28.3.2 [Forcing all links to top-of-page](#) on page 549
- §28.3.3 [Linking to an arbitrary location](#) on page 550

*See also:*

- §28.6 [Linking to other files and other DITA2Go projects](#) on page 553

### 28.3.1 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 anchor or **HyperAnchor** PI 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 §28.5.2 [Specifying HTML options for cross-reference formats](#) on page 552.

### 28.3.2 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 anchor or **HyperAnchor** PI marker location. This is equivalent to setting `NoRef` for all paragraph styles in `[HTMLParaStyles]`; see §28.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 549:

```

[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
```

### 28.3.3 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 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 **HyperAnchor** PI markers.

## 28.4 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 `<a href=...>` tag **DITA2Go** generates for the jump. If you are using framesets that value must be the name of a frame (see §22.13 [Using framesets](#) on page 443), 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 PI marker.](#)

See also:

- §16.7 [Jumping to secondary windows in Help systems](#) on page 262
- §16.8 [Creating pop-up topics for Help systems](#) on page 263

*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 PI marker*

If you need case-by-case handling of jumps to other windows, put a PI marker of type **LinkTarget**, with marker content the name of the window, anywhere before the relevant **HyperJump** PI marker.

## 28.5 Converting DITA cross-reference links to HTML

**DITA2Go** automatically converts DITA cross references to `<a href="...">` links in HTML output. You can specify options for these links.

You can specify settings to prevent some cross references from being converted to links, or to customize cross-reference behavior.

*In this section:*

§28.5.1 [Specifying HTML options for all cross references](#) on page 551

§28.5.2 [Specifying HTML options for cross-reference formats](#) on page 552

*See also:*

§28.6 [Linking to other files and other DITA2Go projects](#) on page 553

### 28.5.1 Specifying HTML options for all cross references

You can specify several processing options that apply to all cross references:

```
[HTMLOptions]
; 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:*

§28.6.1 [Retaining file paths in interfile links](#) on page 553

## 28.5.2 Specifying HTML options for 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 **DITA2Go** 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 <code>&lt;a href=... &gt;</code> tag, so the cross reference does not become a link.
NoRef	Creates the <code>&lt;a href=... &gt;</code> tag, but omits any hash part of the href attribute; for example, <code>file.htm#heading</code> would become just <code>file.htm</code> . 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 <b>DITA2Go</b> 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 §28.2.4 <a href="#">Specifying link properties with macros</a> on page 548), the cross-reference LinkSrc macro prevails. In the macro definition you can use predefined macro variable <code>&lt;\$_linksrc&gt;</code> , which provides the normal href content of the link.

For example:

```
[XrefStyles]
zSectionLink=NoRef
Heading & Page=Text
Page=Delete
```

With these settings, **DITA2Go** 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 §37 [Working with macros](#) on page 679) 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:

§28.2.4 [Specifying link properties with macros](#) on page 548

§28.3.1 [Forcing links to top-of-page for selected paragraph formats](#) on page 549.

## 28.6 Linking to other files and other DITA2Go projects

With default configuration settings, **DITA2Go** 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.

*In this section:*

§28.6.1 [Retaining file paths in interfile links](#) on page 553

§28.6.2 [Enabling links to renamed or relocated files](#) on page 553

*See also:*

§28.7 [Linking to external destinations](#) on page 554

§29 [Providing navigation in HTML](#) on page 555

### 28.6.1 Retaining file paths in interfile links

By default, **DITA2Go** removes paths from interfile links in your DITA files. If your HTML files will be maintained in a directory structure *identical* to the structure used for the DITA files from which they are generated, you must direct **DITA2Go** 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), **DITA2Go** places all HTML files in the project directory, regardless of where the originating DITA 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 DITA files are in a non-flat directory structure (some DITA files are in different directories).
- You move the resulting HTML files to an identical directory structure.

**Note:** You might still get link errors for links between DITA files, especially if you are using the `CodeStore` property (see §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711); 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 DITA 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 DITA files are located.

### 28.6.2 Enabling links to renamed or relocated files

You might run into situations where the names of DITA 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 26.

If you need to rename HTML files *after* **DITA2Go** produces them, you must tell **DITA2Go** 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], **DITA2Go** goes by the original DITA 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.

## 28.7 Linking to external destinations

**DITA2Go** produces HTML links from **HyperLink** PI markers in your document.

To specify a target for a hypertext link (for example, `_blank`):

```
[HTMLOptions]
; URLTarget = name of target to use for all links unless
; otherwise set, default none
URLTarget=_blank
```

To specify an email address:

```
[HTMLOptions]
URLTarget=mailto:name@company.com
```

When a hypertext link specifies an absolute path (which must start with a drive letter), **DITA2Go** prefixes the path with “file:///”. For example:

```
file:///g:/omnisys/ug/out/ugdita2go.pdf
```

For a relative path, **DITA2Go** includes just the text of the destination. For example:

```
../out/ugdita2go.pdf
```

# 29 Providing navigation in HTML

---

To provide a navigation system for HTML output, you can use **DITA2Go** navigation macros, DITA cross references or hypertext links, or a combination of these, to link together the HTML files **DITA2Go** generates from your DITA document. Topics include:

§29.1 [Understanding how navigation links work](#) on page 555

§29.2 [Generating trails of links](#) on page 555

§29.3 [Creating a browse sequence](#) on page 559

*See also:*

§28 [Creating HTML links](#) on page 545

## 29.1 Understanding how navigation links work

**DITA2Go** 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 **DITA2Go** gets the titles and links to use. If you do not split DITA 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 §27.3 [Splitting files](#) on page 526.

## 29.2 Generating trails of links

You can have **DITA2Go** 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:*

§29.2.1 [Understanding trails of links](#) on page 555

§29.2.2 [Specifying whether to include trails of links](#) on page 556

§29.2.3 [Specifying what to include in trails of links](#) on page 556

§29.2.4 [Specifying heading levels for trails of links](#) on page 557

§29.2.5 [Specifying where to display trails of links](#) on page 558

### 29.2.1 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 DITA document that has subheadings can be used as a link in a trail leading to successively lower subheadings. For each trail of links **DITA2Go** 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 §27.3 [Splitting files](#) on page 526.

Trails of links are not compatible with [HTMLOptions]SmartSplit (see §27.3.2.1 [Preventing splits that leave dangling headings](#) on page 527). 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.

## 29.2.2 Specifying whether to include trails of links

To have **DITA2Go** 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, **DITA2Go** 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 §29.2.3 [Specifying what to include in trails of links](#) on page 556)
- Either of the following is true:
  - The format is assigned the [HTMLParaStyles]Split property (see §27.3.1 [Designating split points](#) on page 526)
  - [Trails]SplitTrail=Yes (see §29.2.5 [Specifying where to display trails of links](#) on page 558).

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. **DITA2Go** inserts trails only if MakeTrail=Yes.

## 29.2.3 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 **DITA2Go** 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 §27.5.2.3 [Specifying a title prefix or suffix](#) on page 532).

*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 <em>) 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 &nbsp;&nbsp; of the TrailSep value with <br>.

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.

## 29.2.4 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 DITA 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, **DITA2Go** 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.

## 29.2.5 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>&lt;\$_trail&gt;</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 §27.6 [Inserting HTML code in split and extract files](#) on page 534):

```
[Inserts]
Top=<$_trail>
```

```
SplitBottom=<$_trail>
LastBottom=<$_trail>
```

When **DITA2Go** 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, **DITA2Go** inserts a trail according to the settings you specify in the `[Trails]` section.

## 29.3 Creating a browse sequence

You can use **DITA2Go**-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.

**DITA2Go** also provides predefined macro `<$_top>` for a jump to top-of-page.

*In this section:*

- §29.3.1 [Understanding how browse macros work](#) on page 559
- §29.3.2 [Choosing buttons versus text links for a browse sequence](#) on page 560
- §29.3.3 [Formatting browse-link labels](#) on page 561
- §29.3.4 [Modifying macros `<\$\_prev>`, `<\$\_next>`, and `<\$\_top>`](#) on page 561
- §29.3.5 [Understanding browse keyword scope and default values](#) on page 563
- §29.3.6 [Specifying where to invoke a browse macro](#) on page 564

*See also:*

- §27 [Splitting and extracting files](#) on page 523
- §28 [Creating HTML links](#) on page 545
- §37 [Working with macros](#) on page 679

### 29.3.1 Understanding how browse macros work

The macros described in this section employ the file sequence information from your root map file to simplify setting up navigation tables at the start and end of HTML pages. **DITA2Go** 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:*

- §29.3.1.1 [Understanding how browse macros vary by file position](#) on page 559
- §29.3.1.2 [Understanding how browse macros employ macro variables](#) on page 560

#### 29.3.1.1 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 §29.3.2 [Choosing buttons versus text links for a browse sequence](#) on page 560)
- 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 in the root map
  - the first file produced from the root map
  - the last file produced from the root map.

You can change any of the definitions by changing the macro code assigned to appropriate navigation keywords; see §29.3.4 [Modifying macros <\\$\\_prev>, <\\$\\_next>, and <\\$\\_top>](#) on page 561.

### 29.3.1.2 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 29-1](#) shows the default values of the link destination and link label used in <\$\_prev> and <\$\_next> for each file position.

**Table 29-1 Default destination and label values for browse macros**

Macro	File position in sequence	Destination value	Label value
<\$_prev>	First file produced from root map	<i>None (no destination code)</i>	At Start
	All other files	<\$\$_prevfile>	<\$\$_prevtitle>
<\$_next>	Last file produced from root map	<i>None (no destination code)</i>	At End
	All other files	<\$\$_nextfile>	<\$\$_nexttitle>

[Table 29-2](#) shows the meanings of the macro variables used in browse-macro definitions.

**Table 29-2 Component macro variables for browse macros**

Macro	Macro variable	Description
<\$_prev>	<\$\$_prevfile>	File name of preceding file
	<\$\$_prevtitle>	Title of preceding file
<\$_next>	<\$\$_nextfile>	File name of following file
	<\$\$_nexttitle>	Title of following file
<i>Either macro</i>	<\$\$_currfile>	File name of current file
	<\$\$_currtitle>	Title of current file (the one used in HTML <title> element)

See §27.7 [Referencing split and extract files](#) on page 536 for additional macro variables that refer to file names and titles of split and extracted files.

## 29.3.2 Choosing buttons versus text links for a browse sequence

By default, **DITA2Go** 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, **DITA2Go** navigation macros produce JavaScript code such as the following:

```
<button type="button"
  onclick="javascript:location.href='Destination'">Label</button>
```

When UseNavButtons=No, **DITA2Go** 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 29-1](#) on page 560.



### 29.3.3 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>
```

### 29.3.4 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. **DITA2Go** provides keywords to which you assign macro code for this purpose.

*In this section:*

§29.3.4.1 [Redefining text-link browse macros](#) on page 561

§29.3.4.2 [Redefining button browse macros](#) on page 562

#### 29.3.4.1 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 DITA 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 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
```

### 29.3.4.2 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 DITA 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 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">
  <$$_segendtitle></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.

### 29.3.5 Understanding browse keyword scope and default values

Table 29-3 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 29-3 Scope of [NavigationMacros] keywords**

Scope	Keyword (* = Macro or Button)	Defines:	Default value uses:	
			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
All other split files	Prev*	<\$_prev>	<\$\$_prevfile>	<\$\$_prevtitle>
	Next*	<\$_next>	<\$\$_nextfile>	<\$\$_nexttitle>

Table 29-4 shows the default value **DITA2Go** uses for each browse-macro keyword when the navigation macros produce text links; Table 29-5 shows the default values when macros produce buttons (see §29.3.2 [Choosing buttons versus text links for a browse sequence](#) on page 560).

**Table 29-4 Default values of text-link browse keywords**

Keyword	Default value
PrevMacro	<a href="<\$\$_prevfile>"><\$\$_prevtitle></a>
NextMacro	<a href="<\$\$_nextfile>"><\$\$_nexttitle></a>
StartingPrevFSMacro	At Start
EndingNextFSMacro	At End

**Table 29-5 Default values of button browse keywords**

Keyword	Default value
PrevButton	<code>&lt;button type="button" onclick="javascript:location.href='&lt;\$\$_prevfile&gt;' "&gt; &lt;\$\$_prevtitle&gt;&lt;/button&gt;</code>
NextButton	<code>&lt;button type="button" onclick="javascript:location.href='&lt;\$\$_nextfile&gt;' "&gt; &lt;\$\$_nexttitle&gt;&lt;/button&gt;</code>
StartingPrevFSButton	<code>&lt;button type="button"&gt;At Start&lt;/button&gt;</code>
EndingNextFSButton	<code>&lt;button type="button"&gt;At End&lt;/button&gt;</code>

### 29.3.6 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 §27.6 [Inserting HTML code in split and extract files](#) on page 534 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.

# 30 Mapping text formats to HTML/XML

---

This section shows how to assign HTML elements to paragraph and character formats. Topics include:

- §30.1 [Choosing how to map formats](#) on page 565
- §30.2 [Mapping paragraph formats](#) on page 566
- §30.3 [Mapping character formats](#) on page 569
- §30.4 [Assigning properties to text formats](#) on page 570
- §30.5 [Mapping special characters](#) on page 574
- §30.6 [Mapping fonts](#) on page 576
- §30.7 [Managing typographic elements for HTML or XML](#) on page 579
- §30.8 [Specifying text colors for HTML](#) on page 580
- §30.9 [Configuring preformatted text for HTML/XML](#) on page 581
- §30.10 [Converting footnotes to HTML or XML](#) on page 581
- §30.11 [Converting list formats to HTML \(deprecated\)](#) on page 584

*See also:*

- §31 [Setting up CSS for HTML](#) on page 591

## 30.1 Choosing how to map formats

DITA2Go provides several ways to map formats to HTML, with considerable overlap among methods. You might want to use some or all of the following:

- [Configuration settings](#)
- [Cascading style sheets](#)
- [Output format definitions](#)

*Configuration settings* 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 “bigness” is questionable, especially at the lower end (h4, h5, h6). However, you can use configuration settings to achieve effects not possible with CSS, such as the macro insertion of content. See:

- §30.2 [Mapping paragraph formats](#) on page 566
- §30.3 [Mapping character formats](#) on page 569.

*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 §31 [Setting up CSS for HTML](#) on page 591.

**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.

*Output format definitions* Specify exactly how you want each text feature in your document to look in HTML, by defining output formats in a format configuration file. Format definitions are based on CSS, but are easier to use; see §7 [Configuring output formats](#) on page 109.

## 30.2 Mapping paragraph formats

By default, if you do not explicitly map formats to HTML tags, **DITA2Go** 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 `<hn>` styles, your emphasized text to be tagged `<em>`, and your lists to become real HTML indented lists, without requiring CSS.

*In this section:*

- §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566
- §30.2.2 [Including or excluding paragraph autonumbers](#) on page 567
- §30.2.3 [Designating script paragraph formats](#) on page 568
- §30.2.4 [Stripping paragraph properties](#) on page 568
- §30.2.5 [Providing content for empty paragraphs](#) on page 569

*See also:*

- §30.9 [Configuring preformatted text for HTML/XML](#) on page 581
- §30.10 [Converting footnotes to HTML or XML](#) on page 581
- §30.11 [Converting list formats to HTML \(deprecated\)](#) on page 584

### 30.2.1 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, **DITA2Go** 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 **DITA2Go** 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 PI marker in the paragraph; see §38.3 [Adding attributes with PI markers](#) on page 721. For this example, you would use a PI marker of type **ParaType** with content **bold**.

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 §31.5 [Understanding how CSS affects other options](#) on page 596.

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.

For XML output, see §23.3.1 [Deriving XML tags from format and class names](#) on page 452.

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 §30.2.4 [Stripping paragraph properties](#) on page 568.

If you are producing DITA XML output, see also §24.4.3.2 [Omitting element tags for selected paragraph formats](#) on page 462.

## 30.2.2 Including or excluding paragraph autonumbers

By default, for HTML output **DITA2Go** 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 autonumber in para. For lists, and for
; DocBook and DITA, the default omits it.
ParaFmt = Anum
```

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:

§30.11 [Converting list formats to HTML \(deprecated\)](#) on page 584

§43.2 [Converting autonumbers for database systems](#) on page 780

### 30.2.3 Designating script paragraph formats

A paragraph tagged as script in [ParaTags] (see §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566) 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 §30.2.4 [Stripping paragraph properties](#) on page 568.

### 30.2.4 Stripping paragraph properties

You can designate text in your DITA document that you do not want fully converted to HTML. For example, you can include material pre-written in HTML, and direct **DITA2Go** 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
```

*Comment* Use Comment to cause a paragraph to appear only as a comment in the generated HTML source code. **DITA2Go** substitutes <!-- and --> tags for the <p> and </p> tags, unless

you have also assigned a `script` tag to the paragraph format in `[ParaTags]`; see §30.2.3 [Designating script paragraph formats](#) on page 568.

- NoTags* Use `NoTags` to suppress all tags between `<p>` and `</p>` (such as `<font>`, `<b>`, `<i>`, and so forth) in the generated HTML for the paragraph. Only the `<p>` tags themselves and the paragraph content are included in the output.
- NoPara* Use `NoPara` to suppress only the `<p>` tags in the output. You might want to do this when either of the following is true:
- The paragraph will be part of a **DITA2Go** macro that already supplies `<p> ... </p>`.
  - You are generating XML instead of HTML; see §23.3.2 [Eliminating HTML attributes and tags for generic XML](#) on page 452. If you are producing DITA XML output, also see §24.4.3.2 [Omitting element tags for selected paragraph formats](#) on page 462.
- NoWrap* Use `NoWrap` to suppress `\n` line breaks and preserve leading spaces in preformatted text. This property has the same effect as `[HTMLOptions]NoWrap`, but applied at the paragraph format level; see §22.6.3 [Suppressing line breaks in HTML and XML output](#) on page 439.
- Raw* Use `Raw` to insert straight HTML code wherever you want it to appear in your document. **DITA2Go** embeds the content of the paragraph in the output without generating HTML tags, and without processing any macro invocations the content might include.

### 30.2.5 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.

### 30.2.6 Eliminating unwanted paragraphs

To prevent text from appearing in HTML output, you can do the following:

1. Use a special `@outputclass` for all instances of the text in your document.
2. In the configuration file, assign property `Delete` to the resulting 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 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 **DITA2Go** macros; see §37.3.5 [Creating macro variables from paragraph content](#) on page 692.

## 30.3 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, **DITA2Go** lets you specify any tag, to permit XML markup and CSS `span` class assignments. However, only valid HTML tags have an effect in **DITA2Go** 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 PI marker in the inline element; see §38.3 [Adding attributes with PI markers](#) on page 721. For this example, you would use a PI 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 §31.7.3 [Mapping character formats to tags or span classes](#) on page 602.

*Suppress tags* To eliminate style tags entirely, map the character format to nothing:

```

[CharTags]
CharFmt =

```

*See also:*

§23.3.1 [Deriving XML tags from format and class names](#) on page 452

§31.7.3 [Mapping character formats to tags or span classes](#) on page 602

## 30.4 Assigning properties to text formats

You can override some paragraph and character format properties directly, and you can assign additional properties to paragraph and character formats.

*In this section:*

§30.4.1 [Understanding where to specify format property overrides](#) on page 570

§30.4.2 [Overriding paragraph alignment and size properties](#) on page 573

§30.4.3 [Overriding properties added by typographic elements](#) on page 573

§30.4.4 [Overriding properties specified in font tags](#) on page 573

### 30.4.1 Understanding where to specify format property overrides

In prior versions of **DITA2Go**, 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 30-1](#) lists all the properties alphabetically, and shows the sections in which they are valid: **Para** for `[HTMLParaStyles]` and **Char** for `[HTMLCharStyles]`.

**Table 30-1 HTML properties for paragraph and character formats**

Format property	Purpose	Para	Char	Ref.
Abbr	Gets value for abbr attribute from [StyleCellAbbr]	X		35.2.2.2
AbbrVal	Make content into abbr for table cell	X		35.2.3
ALink	Uses content for ALink Name property of ALink object	X		16.6.4.2
Alt	Makes content into alt attribute for next <img>	X		34.2.2
Anum	Includes autonumber in format	X		30.2.2
Axis	Gets value for axis attribute from [StyleCellAxis]	X		35.2.2.2
AxisVal	Makes content into axis for table cell	X		35.2.3
Bold	Encloses text in this format in <b> . . . </b>	X	X	30.4.3
CellAttribute	Applies attributes in [StyleCellAttribute] to enclosing cell	X		33.4.6
Center	Centers text between left and right margins	X		30.4.2
CodeAfter	Puts code from [ParaStyleCodeAfter] or [CharStyleCodeAfter] after closing tag	X	X	37.9.3
CodeAfterAnum	Puts code from [AnumCodeAfter] after autonumber	X		37.9.3
CodeBefore	Puts code from [ParaStyleCodeBefore] or [CharStyleCodeBefore] before opening tag	X	X	37.9.3
CodeBeforeAnum	Puts code from [AnumCodeBefore] before autonumber	X		37.9.3
CodeEnd	Puts code from [ParaStyleCodeEnd] or [CharStyleCodeEnd] before closing tag	X	X	37.9.3
CodeReplace	Replaces content with code from [ParaStyleCodeReplace] or [CharStyleCodeReplace]	X	X	37.9.3
CodeStart	Puts code from [ParaStyleCodeStart] or [CharStyleCodeStart] after opening tag	X	X	37.9.3
CodeStore	Stores content in macro variable named in [StyleCodeStore]	X		37.3.5.2
ColGroup	Marks enclosing cell as a header cell that starts a column group	X		35.2.2.2
ColorN	Makes text color N, where N is in the range 1 - 254	X	X	30.4.4
Comment	Makes paragraph a comment, replacing tags	X	X	30.2.4
Config	Makes content act as a configuration override for all outputs	X		42.3
Contents	Includes content in this format in the TOC	X		16.4.2
CSSReplace	Gets code for CSS from [ParaStyleCSS] or [CharStyleCSS]	X	X	31.8.3
Delete	Omits content from text output	X	X	30.2.6
DListDD	Uses dd instead of dt for items in dl lists	X		30.11.2.1
DropDown	Content is a block to be expanded; bracket with macros	X		16.9.3.1
DropDownBlock	Content is a block to be expanded	X		16.9.3.1
DropDownEnd	Content is last paragraph in expandable block	X		16.9.3.1
DropDownLink	Content is a link to expandable text	X	X	16.9.3.1
DropDownStart	Content is a link, next content is an expandable block	X		16.9.3.1
ExtrDisable	Turns off extract processing	X		27.4.1
ExtrEnable	Turns on extract processing	X		27.4.1
ExtrEnd	Paragraph ends an extract, but is not part of the extract	X		27.4.2.1
ExtrFinish	Paragraph is the last item in an extract	X		27.4.2.1
ExtrStart	Paragraph begins an extract	X		27.4.2.1
Figure	Uses paragraph for anchor tag to ensure wrapping image in <fig>	X		24.7.2
FileName	Uses content to name split files	X		43.3.3.1

**Table 30-1 HTML properties for paragraph and character formats (continued)**

Format property	Purpose	Para	Char	Ref.
GlossTerm	Uses content for a glossary term in JavaHelp	X		<a href="#">20.7.2</a>
HTMConfig	Makes content act as a configuration override for HTML output	X		<a href="#">42.3</a>
Ital	Italics: encloses text in this format in <code>&lt;i&gt;...&lt;/i&gt;</code>	X	X	<a href="#">30.4.3</a>
Left	Aligns text with left margin	X		<a href="#">30.4.2</a>
LEnd	Non-list format that ends any prior lists	X		<a href="#">30.11.2.1</a>
LFirst	Content in this format starts a list	X		<a href="#">30.11.2.1</a>
LinkClass	Makes content into CSS <code>class</code> attribute value	X		<a href="#">34.3.2</a>
LinkSrc	Puts code from <code>[StyleLinkSrc]</code> in <code>href</code> attribute	X	X	<a href="#">37.9.3</a>
LinkTitle	Makes content into <code>title</code> attribute value	X		<a href="#">34.3.2</a>
ListN	<code>List1</code> - <code>List12</code> specify different list styles	X		<a href="#">30.11.2.1</a>
LLevelN	<code>LLevel1</code> - <code>LLevel30</code> specify nesting levels	X		<a href="#">30.11.2.1</a>
LNest	Nests in an enclosing list	X		<a href="#">30.11.2.1</a>
Longdesc	Makes content into <code>longdesc</code> attribute value	X		<a href="#">34.2.2</a>
Meta	Makes content a <code>&lt;meta content=...&gt;</code> attribute value	X		<a href="#">27.5.3</a>
NoAnum	Excludes autonumber from non-list formats	X		<a href="#">30.2.2</a>
NoColID	Prevents assignment of <code>id</code> for <code>ColIDs</code> (enabled in <code>[Tables]</code> )	X		<a href="#">35.2.2.2</a>
NoColor	Omits <code>&lt;font color=...&gt;</code> for this format	X	X	<a href="#">30.4.4</a>
NoContLink	Suppresses linkage for the corresponding TOC item in HTML Help	X		<a href="#">18.9.5</a>
NoCSS	Omits any CSS entry for this format	X	X	<a href="#">31.8.3</a>
NoFig	Uses paragraph for anchor tag to prevent wrapping image in <code>&lt;fig&gt;</code>	X	X	<a href="#">24.7.2</a>
NoPara	Eliminates only paragraph tags, for use in macros	X	X	<a href="#">30.2.4</a>
NoRef	Forces links to top of page by suppressing part of link after file name	X	X	<a href="#">28.3.1</a>
NoSize	Omits <code>size</code> attribute from <code>&lt;font&gt;</code> tags for this format	X	X	<a href="#">30.4.4</a>
NoSplit	Prevents format from interfering with <code>SmartSplit</code>	X		<a href="#">27.3.2.2</a>
NoTags	Suppresses attributes and omits any tags between <code>&lt;p&gt;</code> and <code>&lt;/p&gt;</code>	X	X	<a href="#">30.2.4</a>
NoWrap	Suppresses <code>\n</code> line breaks and preserves leading spaces	X		<a href="#">30.2.4</a>
Plain	Turns off <code>Bold</code> , <code>Ital</code> , <code>Uline</code> , and <code>Strike</code> format properties	X	X	<a href="#">30.4.3</a>
ParaLinkClass	Links have a class assigned in <code>[StyleParaLinkClass]</code>	X		<a href="#">28.2.2.2</a>
Raw	Suppresses all tags to allow macro inclusions between document elements	X	X	<a href="#">30.2.4</a>
Right	Aligns text with right margin	X		<a href="#">30.4.2</a>
RowAttribute	Attributes in <code>[StyleRowAttribute]</code> are applied to enclosing row	X		<a href="#">33.4.5</a>
RowGroup	Marks enclosing cell as a header cell that starts a row group	X		<a href="#">35.2.2.2</a>
Scope	Gets value for <code>scope</code> attribute from <code>[StyleCellScope]</code>	X		<a href="#">35.2.2.2</a>
SizeN	<code>Size1</code> - <code>Size7</code> sets font <code>size</code> attribute to 1 through 7, corresponding to maximum point sizes 8, 10, 14, 20, 28, or 36	X		<a href="#">30.4.2</a>
Span	Causes assignment of <code>ColSpanID</code> or <code>RowSpanID</code> , as enabled in <code>[Tables]</code>	X		<a href="#">35.2.2.2</a>
Split	Starts a new HTML page	X		<a href="#">27.3.1</a>
Strike	Strikethrough: enclose text in this format in <code>&lt;strike&gt;...&lt;/strike&gt;</code>	X	X	<a href="#">30.4.3</a>
Summary	Makes content into summary for table tag	X		<a href="#">34.4.3.2</a>
TableBody	Forces containing cell tag to <code>td</code> instead of <code>th</code>	X		<a href="#">35.2.2.4</a>
TableHead	Forces containing cell tag to <code>th</code> instead of <code>td</code>	X		<a href="#">35.2.2.4</a>
TableTitle	Makes content into <code>title</code> attribute for table	X		<a href="#">34.4.3.2</a>

**Table 30-1 HTML properties for paragraph and character formats (continued)**

Format property	Purpose	Para	Char	Ref.
TextStore	Stores content in macro variable named in [StyleTextStore]	X		<a href="#">37.3.5.1</a>
Title	Content in this format becomes HTML page title	X		<a href="#">22.4.5</a>
Trail	Includes content in breadcrumb trail of links	X		<a href="#">29.2.3</a>
ULine	Underline: encloses text in this format in <u>...</u>	X	X	<a href="#">30.4.3</a>
Window	Opens topic in window named in [StyleWindow] for HTML Help	X		<a href="#">18.8.3.1</a>

### 30.4.2 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 §30.6.2 [Mapping font sizes](#) on page 577.

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 §31.5 [Understanding how CSS affects other options](#) on page 596.

### 30.4.3 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 (<b>, <i>, <u>, and <strike> tags) from selected paragraph or character formats:

```
[HTMLParaStyles] or [HTMLCharStyles]
Format = Plain
```

### 30.4.4 Overriding properties specified in font tags

To override paragraph or character properties added as <font> attributes:

```
[HTMLParaStyles] or [HTMLCharStyles]
; Color1 - Color254 color text as defined in [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 `<font>` tags used within a paragraph or character span for its default style, provided you are including `<font>` tags. Using CSS turns `<font>` tags off by default; see §30.6 [Mapping fonts](#) on page 576 and §31.5 [Understanding how CSS affects other options](#) on page 596.

The `Colornnn` properties use color numbers to assign text colors to paragraph or character formats; see §30.8 [Specifying text colors for HTML](#) on page 580.

## 30.5 Mapping special characters

Special-character handling is not a strong point of HTML. **DITA2Go** 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 **DITA2Go** from mapping special characters.

*In this section:*

§30.5.1 Converting Western European accented characters on page 574

§30.5.2 Mapping individual special characters on page 574

§30.5.3 Avoiding use of special characters in URIs on page 576

§30.5.4 Preventing character mapping on page 576

*See also:*

§22.4.3 Specifying character encoding for HTML on page 434

§22.14.2 Replacing high ASCII characters for W3C validation on page 445

§23.2.3 Specifying character encoding for generic XML on page 450

### 30.5.1 Converting Western European accented characters

**Mit freundlichen Grüßen.** DITA2Go 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 **DITA2Go** replaces an accented character with the corresponding non-accented character; see §31.7.1 [Understanding CSS class name restrictions](#) on page 600.

### 30.5.2 Mapping individual special characters

To force a mapping different from the **DITA2Go** 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
- 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 **DITA2Go** treats as wildcards unless you disable this feature; see §4.1.10 [Specifying how to treat cases, spaces, and wildcards](#) on page 73
- a high ASCII character (decimal code 128 through 159).

Table 30-2 shows the Unicode or other hexadecimal (and in some cases, decimal) value you can specify to the left of the equals sign.

**Table 30-2 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 **DITA2Go** macro references.

When you supply a string rather than a character code, **DITA2Go** 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, &lt;.

*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).

*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, **DITA2Go** 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;
```

### 30.5.3 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. **DITA2Go** 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.

### 30.5.4 Preventing character mapping

You can prevent **DITA2Go** from mapping high ASCII characters to entity references:

```
[HTMLOptions]
Encoding = None
```

However, this option does not produce valid HTML; see §22.14 [Passing W3C validation tests](#) on page 445.

*See also:*

§22.4.3 [Specifying character encoding for HTML](#) on page 434

## 30.6 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:*

§30.6.1 [Specifying a default font and size](#) on page 577

§30.6.2 [Mapping font sizes](#) on page 577

§30.6.3 [Including or excluding font tags](#) on page 578

§30.6.4 [Excluding face and size attributes from font tags](#) on page 578

§30.6.5 [Accommodating browser font-rendering differences](#) on page 579

## 30.6.1 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 §31.5 [Understanding how CSS affects other options](#) on page 596.

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 `<font>` tags either, you will get whatever fonts a browser specifies as defaults.

## 30.6.2 Mapping font sizes

For CSS, **DITA2Go** shows the font size just as it is defined for the format. For HTML itself, **DITA2Go** must convert point size to an HTML size number, 2 through 7. To modify the way **DITA2Go** 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 `<font size="3">` for 14-point text, by raising the start of the size 4 range to 15-point text.

The [FontSizes] settings determine how **DITA2Go** converts from points to HTML size numbers, regardless of any other settings. Also see §30.4.2 [Overriding paragraph alignment and size properties](#) on page 573.

To change CSS entries from points to other size units, see §31.8.2 [Specifying CSS size values and units of measurement](#) on page 607.

### 30.6.3 Including or excluding font tags

Older versions of Internet Explorer contain a defect in how <font> tags are handled. If your HTML output includes a <font> 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 <font> tag and shows the correct character, in compliance with the W3 specification.

By default, when you use CSS, **DITA2Go** does not include <font> tags in HTML output. However, you might need additional <font> tags in some circumstances. For example:

- If you use an OpenType or TrueType font, some browsers require <font> tags to correctly display content in these fonts; see §30.6.5 [Accommodating browser font-rendering differences](#) on page 579.
- If you are producing JavaHelp or Oracle Help, you might need <font> tags to get around viewer deficiencies; see §20.3.8 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 394.

To turn on <font> 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, **DITA2Go** turns on <font> tags by default; see §31.5 [Understanding how CSS affects other options](#) on page 596. If you do use CSS, you can create <span> tags instead, with a single setting for each character format; see §31.7.3 [Mapping character formats to tags or span classes](#) on page 602.

If you do not use CSS, and you are not using <font> tags either, you will get whatever fonts a browser specifies as defaults.

### 30.6.4 Excluding face and size attributes from font tags

To suppress the face attribute in <font> 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 <font> 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 §20.3.8 [Coping with JavaHelp / Oracle Help viewer limitations](#) on page 394.

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 §30.6.5 [Accommodating browser font-rendering differences](#) on page 579.

To suppress the size attribute in `<font>` tags:

```
[HTMLOptions]
;UseFontSize = Yes (default, allow size attribute in <font>) or No
UseFontSize = No
```

### 30.6.5 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 `<font>` 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 `<font>` 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 **DITA2Go** to use `<font>` 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.

## 30.7 Managing typographic elements for HTML or XML

By default, for HTML output **DITA2Go** provides typographic elements for 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 `<b>` elements as well as the code for the paragraph tag.

*In this section:*

§30.7.1 [Deciding whether to suppress typographic elements](#) on page 579

§30.7.2 [Choosing how to treat typographic elements](#) on page 579

### 30.7.1 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.

For XML output, including DITA XML and DocBook XML, the default is to suppress all typographic elements.

### 30.7.2 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.; Both "over" for overline and
"chbar" for change bar can be used
; as pseudotags here.

```

You can choose to:

[Suppress all typographics](#)

[Replace typographics with other tags.](#)

*Suppress all  
typographics*

To eliminate all typographic elements:

```

[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 character or paragraph format or were applied as an override. This setting, UseTypographicElements=No, cannot be overridden by any other settings in section [Typographics]. This is the appropriate value for DITA XML output; see §24.4.4 [Mapping character formats to DITA inline elements](#) on page 465.

*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 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.

## 30.8 Specifying text colors for HTML

The best way to adjust colors for HTML output is to assign colors to output formats; see:

§7.6.5 [Specifying inline properties for paragraph and character formats](#) on page 123

§7.6.6 [Specifying block properties for paragraph formats](#) on page 124

Those methods use CSS. Use the method described in this section only if you cannot use CSS.

You can specify colors for both character and paragraph formats. Text color is set in CSS; and also in <font> tags, if you leave them enabled; see §30.6.3 [Including or excluding font tags](#) on page 578.

To change the color of text in a paragraph or character format:

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 §22.7.1 [Numbering and defining text colors](#) on page 440.

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 [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
```

## 30.9 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.

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`, **DITA2Go** ignores all line breaks in text mapped to preformatted elements, including 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:

§22.6.3 [Suppressing line breaks in HTML and XML output](#) on page 439

## 30.10 Converting footnotes to HTML or XML

If you are not using CSS (see §31.1 [Deciding whether to use CSS](#) on page 591), **DITA2Go** sets the `type` attribute of the `<ol>` used for the footnotes. If the document has



a custom footnote type, **DITA2Go** uses a `<div>` with the class instead of `<ol>`, and a `<p>` instead of `<li>`, and writes out the symbols.

*In this section:*

- §30.10.1 [Configuring and placing footnotes](#) on page 582
- §30.10.2 [Eliminating links to jump footnotes](#) on page 583
- §30.10.3 [Using list tags or `<div>` and `<p>` tags for jump footnotes](#) on page 583
- §30.10.4 [Formatting jump footnote text with macros](#) on page 583

*See also:*

- §31.7.5 [Assigning CSS classes to text and table footnotes](#) on page 603

## 30.10.1 Configuring and placing footnotes

**DITA2Go** provides the following options for placement of footnotes from your DITA 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.

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 §27 <a href="#">Splitting and extracting files</a> on page 523), 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 <code>&lt;br /&gt;&lt;br /&gt;&lt;hr /&gt;</code> .
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
```

## 30.10.2 Eliminating links to jump footnotes

By default, **DITA2Go** creates a link for each reference to a Jump footnote (see §30.10.1 [Configuring and placing footnotes](#) on page 582).

To eliminate links to footnotes:

```
[HTMLOptions]
; NoFootnoteLinks = No (default) or Yes (eliminate links to footnotes)
NoFootnoteLinks = Yes
```

When `NoFootnoteLinks=Yes` and `Footnotes=Jump` (see §30.10.1 [Configuring and placing footnotes](#) on page 582), footnotes appear where and how specified, but references to them do not contain active links.

## 30.10.3 Using list tags or <div> and <p> tags for jump footnotes

By default, **DITA2Go** uses list tags for Jump footnotes in both text and tables (see §30.10.1 [Configuring and placing footnotes](#) on page 582). For numbered footnotes, **DITA2Go** supplies Arabic numerals or Roman numerals; see §8.5.4 [Defining footnote numbering](#) on page 150. For alphabetic footnotes, if the quantity of footnotes per page exceeds the length of the alphabet, **DITA2Go** 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 §31.7.5 [Assigning CSS classes to text and table footnotes](#) on page 603.

## 30.10.4 Formatting jump footnote text with macros

By default, **DITA2Go** removes any paragraph start/end coding within a footnote. However, for Jump footnotes (see §30.10.1 [Configuring and placing footnotes](#) on page 582) 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 `<ul>` and `<li>` coding for the bulleted paragraphs.

**Note:** In HTML the convention is not to use `</li>`, because this closing tag can create extra unwanted spacing in some browsers. However, `</li>` is required in XHTML and XML.

If you code numbered footnotes as `<li>` items, they should be in an `<ol>` block, which provides the numbering. Numbering restarts at 1 for each HTML page. See §30.11.2 [Converting list formats to HTML list styles](#) on page 585.

## 30.11 Converting list formats to HTML (*deprecated*)

Lists, especially nested lists, are a challenge to format correctly for HTML output. The settings described in this section are deprecated in favor of CSS, or list formats defined in format configuration files; see §7 [Configuring output formats](#) on page 109.

*In this section:*

§30.11.1 [Understanding the problem with HTML lists](#) on page 584

§30.11.2 [Converting list formats to HTML list styles](#) on page 585

§30.11.3 [Indenting list items](#) on page 589

§30.11.4 [Converting list formats to HTML/XML paragraphs](#) on page 589

*See also:*

§7.6.7.1 [Assigning properties to list formats for HTML list styles](#) on page 125

### 30.11.1 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; }
```

**DITA2Go** 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;
}
```

**DITA2Go** uses separate rules for `ol` and `ul` because some viewers (notably the JavaHelp viewer) do not follow CSS cascading rules correctly.

### 30.11.2 Converting list formats to HTML list styles

HTML list tags are far more restrictive than straight CSS. However, if either of the following is true, you should be able to successfully convert list formats to HTML list styles:

- The list formats defined for your document do not have complex autonumbers
- You can allow any list formats that have complex autonumbers to become `<p>` items in the output instead.

In most cases browsers will reformat the list styles unaided, and they will come out as you expect. However, see §30.11.3 [Indenting list items](#) on page 589 for ways you might have to modify CSS properties to line up indents.

**Note:** Unless you specify XHTML as the output type, **DITA2Go** does not generate `</li>` closing tags, because browsers tend to space poorly when `</li>` is present.

*In this section:*

- §30.11.2.1 [Specifying HTML list styles \(deprecated\)](#) on page 585
- §30.11.2.2 [Converting lists with multiple paragraph formats](#) on page 586
- §30.11.2.3 [Converting nested lists](#) on page 587
- §30.11.2.4 [Converting dictionary lists](#) on page 588
- §30.11.2.5 [Omitting CSS class attributes from list entries](#) on page 588
- §30.11.2.6 [Including or excluding the type list attribute](#) on page 588

#### 30.11.2.1 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:** **DITA2Go** 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 §30.11.2.2 [Converting lists with multiple paragraph formats](#) on page 586.
- Assign LEnd to each non-list paragraph format that immediately follows a list; see §30.11.2.2 [Converting lists with multiple paragraph formats](#) on page 586.
- Assign LNest and LLevel $n$  to each paragraph format in a list that is nested inside another list; see §30.11.2.3 [Converting nested lists](#) on page 587.

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

```

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, **DITA2Go** applies the same class name used in the first <li> item in a list to the <ol> or <ul> 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 §30.11.2.2 [Converting lists with multiple paragraph formats](#) on page 586.
- Nested lists; see §30.11.2.3 [Converting nested lists](#) on page 587.
- Dictionary-style lists; see §30.11.2.4 [Converting dictionary lists](#) on page 588.

### 30.11.2.2 Converting lists with multiple paragraph formats

If you use a different paragraph format for the first item in a list, and perhaps also for the last item, specify the appropriate List $n$  style for all of the paragraph formats; and also specify the following properties:

	<pre>[HTMLParaStyles] ; LFirst specifies a style that starts a list ; LEnd specifies a non-list style that ends any prior lists</pre>
<i>LFirst starts a list</i>	Assign LFirst to the format that starts a list. If you do not assign LFirst to a format that can start a list, <b>DITA2Go</b> thinks an item in that format is being continued after a non-list item; if that is not the case, <b>DITA2Go</b> might be using a value that was never set.
<i>LEnd comes after a list</i>	Do not assign LEnd to the format that ends a list; instead, assign LEnd to each paragraph format that can occur immediately <i>after</i> the end of a numbered list in your DITA 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 <i>end</i> a list, as opposed to being <i>included in</i> a list. To avoid unwanted left indents you must assign LEnd to <i>Body</i> , to all the headings, to figure titles and table anchors, and so forth.

### 30.11.2.3 Converting nested lists

If you used nested lists in your DITA document, you must assign the following properties to inner list formats:

	<pre>[HTMLParaStyles] ; LNest specifies a style that nests in an enclosing list ; LLevel specifies the nesting level to use, 1-30</pre>
<i>Make each level a different list type</i>	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 LLevel2 and LLevel1. You can nest quite deeply and still retain the structure, if you apply these properties correctly.

Suppose your DITA 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 LLevel2 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 LLevel2
AlphaSub = List4 LLevel2 LLevel2
```

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. **DITA2Go** 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.

### 30.11.2.4 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, **DITA2Go** 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
```

### 30.11.2.5 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 §31.5 [Understanding how CSS affects other options](#) on page 596.

### 30.11.2.6 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; **DITA2Go** 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, **DITA2Go** omits the `type` attribute from list wrappers `ol` and `ul`. To add the `type` attribute to list wrappers:

```
[CSS]
; NoAttribLists = Yes (default, omit type attribute from list tags),
; or No (include type, start, and value attributes in list tags)
NoAttribLists = No
; 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 No for **Mif2Go**.

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.



### 30.11.3 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:

### §30.11.3.1 Adjusting the second-line list indent in CSS on page 589

§30.11.3.2 Inserting spaces between first-line list autonumber and text on page 589

### 30.11.3.1 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 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 `<ol>` or `<ul>` tags, you must deduct from your CSS indent the amount of indent applied automatically by the browser, or you will see your `<li>` 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 **DITA2Go** supports this method and provides rudimentary JavaScript (see §31.6.1 [Selecting a CSS file at run time](#) on page 597), this is not a standards-friendly approach.

### 30.11.3.2 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 `nbspace`-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 §37.3 [Using macro variables](#) on page 687), but synchronizing the count with the numbering would be challenging.

### 30.11.4 Converting list formats to HTML/XML paragraphs

**DITA2Go** can map list formats to <p> items. If you allow **DITA2Go** to create a CSS file for your document (see §31 [Setting up CSS for HTML](#) on page 591), **DITA2Go** uses CSS to supply 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.
- 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.

See §7.6.6 [Specifying block properties for paragraph formats](#) on page 124 for the properties you can assign to list formats.

# 31 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:

- §31.1 [Deciding whether to use CSS](#) on page 591
- §31.2 [Understanding how to use CSS](#) on page 591
- §31.3 [Understanding how DITA2Go generates CSS](#) on page 592
- §31.4 [Specifying CSS file and link options](#) on page 593
- §31.5 [Understanding how CSS affects other options](#) on page 596
- §31.6 [Linking to alternate CSS files](#) on page 597
- §31.7 [Assigning CSS classes](#) on page 599
- §31.8 [Customizing CSS properties](#) on page 606

## 31.1 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 **DITA2Go** to use.
- Have **DITA2Go** 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 `<font>` 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 **DITA2Go** 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 document; see §31.4 [Specifying CSS file and link options](#) on page 593.

To look at your page in different browsers, using a Web-based method:

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

*See also:*

- §31.5 [Understanding how CSS affects other options](#) on page 596

## 31.2 Understanding how to use CSS

If you are not familiar with CSS, here are some good starting points, tutorials, and reference sites:

<http://websitesetips.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)

## 31.3 Understanding how DITA2Go generates CSS

**DITA2Go** generates CSS based on the formats to which you map the DITA elements in your 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. **DITA2Go** 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 §31.4 [Specifying CSS file and link options](#) on page 593. 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, **DITA2Go** cannot add them to the CSS; you have to do that yourself.

**DITA2Go** gets the formatting information from the formats defined for your document via the [Templates]Formats file chain. **DITA2Go** provides reasonable starting defaults so you get good-looking results out of the box. You can change them to your taste in numerous ways; see §7 [Configuring output formats](#) on page 109.

By default, the first time you convert a document to HTML or XML, or generate HTML-based Help, **DITA2Go** creates a style sheet for the output. **DITA2Go** creates a new CSS file that contains all the paragraph and character format names from your document, based on the following:

- whatever CSS classes you assign to those formats in [ParaClasses] or [CharClasses]; see:
  - §31.7.2 [Mapping paragraph formats to CSS classes](#) on page 601
  - §31.7.3 [Mapping character formats to tags or span classes](#) on page 602
- whatever tags you set for those formats in [ParaTags] and [CharTags]; see:
  - §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566
  - §30.3 [Mapping character formats](#) on page 569.

Absent explicit CSS settings in [ParaClasses], [CharClasses], [ParaTags], or [CharTags], **DITA2Go** bases CSS class names on your format names, possibly reduced to fit CSS naming rules for class names; see §31.7.1 [Understanding CSS class name restrictions](#) on page 600.

See §31.7 [Assigning CSS classes](#) on page 599.

## 31.4 Specifying CSS file and link options

*In this section:*

§31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593

§31.4.2 [Designating and locating a CSS file](#) on page 595

§31.4.3 [Directing DITA2Go to generate a CSS file](#) on page 595

§31.4.4 [Understanding effects of the older Stylesheet setting](#) on page 596

### 31.4.1 Specifying CSS options in a DITA2Go configuration file

To specify CSS options in a **DITA2Go** 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 DITA2Go 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 31-2 [CSS-dependent default values of options](#) on page 597.

**Note:** If you have been using [HtmlOptions]Stylesheet to specify CSS file options, see §31.4.4 [Understanding effects of the older Stylesheet setting](#) on page 596. The Stylesheet setting is deprecated in favor of the [CSS] settings listed in this section.

*Direct **DITA2Go**  
to use CSS*

To direct **DITA2Go** to use CSS for your output:

```
[CSS]
; UseCSS = Yes (default) or No
UseCSS=Yes
```

When UseCSS=Yes, by default **DITA2Go** 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 §31.5 [Understanding how CSS affects other options](#) on page 596.

<i>Include class attributes</i>	WriteClassAttributes values have the following effects:	
	Yes	<b>DITA2Go</b> includes CSS class attributes in the paragraph tags in your output; see §31.3 <a href="#">Understanding how DITA2Go generates CSS</a> on page 592.
	No	Class attributes are not included in paragraph tags. Use this setting for XML output when [CSS]ClassIsTag=Yes, the default for XML; see §23.3.1 <a href="#">Deriving XML tags from format and class names</a> on page 452.
<i>Designate a CSS file</i>	CssFileName designates the CSS file <b>DITA2Go</b> 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 §31.4.2 <a href="#">Designating and locating a CSS file</a> on page 595.	
<i>Create a CSS file</i>	WriteCssStylesheet values have the following effects:	
	Once	<b>DITA2Go</b> creates a new CSS file based on your formats, but only if no CSS file of the name designated by CssFileName is already present in the project directory. This is the default <b>DITA2Go</b> puts in place at set-up. Specify Once to get a starting CSS file that you can tweak manually. See §31.4.3 <a href="#">Directing DITA2Go to generate a CSS file</a> on page 595.
	Always	<b>DITA2Go</b> creates a new CSS file based on your 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 §31.4.3 <a href="#">Directing DITA2Go to generate a CSS file</a> on page 595. You can specify additional settings to govern what <b>DITA2Go</b> includes in a CSS file; see §31.8.3 <a href="#">Overriding styles in DITA2Go-generated CSS files</a> on page 608.
	Never	<b>DITA2Go</b> does not create a new CSS file, nor overwrite an existing file. When UseCSS=Yes, <b>DITA2Go</b> 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 §31.4.2 <a href="#">Designating and locating a CSS file</a> on page 595.
<i>Link to a CSS file</i>	WriteCssLink values have the following effects:	
	Yes	<p>If CssBrowserDetect is not present, <b>DITA2Go</b> includes in the &lt;head&gt; 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>&lt;link rel="stylesheet" href="local.css" type="text/css"&gt;</pre> <p>For XML:</p> <pre>&lt;?xml:stylesheet href="local.css" type="text/css"   charset="UTF-8"?&gt;</pre> <p>If CssBrowserDetect is present, instead of the simple link <b>DITA2Go</b> includes the macro assigned to CssBrowserDetect in the &lt;head&gt; element. See §31.6.1 <a href="#">Selecting a CSS file at run time</a> on page 597.</p>
	No	<b>DITA2Go</b> 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 §31.6.1 <a href="#">Selecting a CSS file at run time</a> on page 597.

*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 §31.6.1 [Selecting a CSS file at run time](#) on page 597.

### 31.4.2 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, **DITA2Go** changes them to forward slashes before writing the path to your HTML output files. The default value of `CssPath` is the output directory; see §44.8 [Placing CSS or XSL files for assembly](#) on page 800. You can have **DITA2Go** 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, **DITA2Go** creates a CSS file named `local.css` and places it in the project directory:

- You have indicated that you want **DITA2Go** 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`.

### 31.4.3 Directing DITA2Go to generate a CSS file

The first time you convert files for a project, if you intend to use CSS, probably you will want **DITA2Go** to generate a new CSS file, so you can use a style sheet that contains the equivalents of your format settings. You should also specify a name for the CSS file (see §31.4.2 [Designating and locating a CSS file](#) on page 595); for example:

```
[CSS]
UseCSS=Yes
WriteCssStylesheet=Once
CssFileName=MyStyles.css
```

When `WriteCssStylesheet=Once`, **DITA2Go** 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 formats will *not* be reflected in the CSS file.



*Force a new CSS, update CSS from formats*

To force **DITA2Go** 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 **DITA2Go** generate a CSS file each time; then any changes you make to your 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 §31.8.3 [Overriding styles in DITA2Go-generated CSS files](#) on page 608

### 31.4.4 Understanding effects of the older Stylesheet setting

Prior versions of **DITA2Go** 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 §31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593. 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 31-1](#). If both are present, the [CSS] options prevail.

**Table 31-1 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

## 31.5 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 changes some default values](#)

[Not using CSS changes other default values.](#)

*Using CSS  
changes some  
default values*

When UseCSS=Yes, default values are reversed for the [Graphics] and [HtmlOptions] settings listed in [Table 31-2](#). This removes most HTML that can interfere with CSS settings.

*Not using CSS  
changes other  
default values*

When UseCSS=No, default values are reversed for the [CSS] settings listed in [Table 31-2](#).

**Table 31-2 CSS-dependent default values of options**

Section	Option	Default value of option when:		
		UseCSS=Yes	UseCSS=No	Ref.
[CSS]	LinkClassIsParaClass	Yes	No	<a href="#">28.2.2.2</a>
	NoClassLists	No	Yes	<a href="#">30.11.2.5</a>
	WriteClassAttributes	Yes	No	<a href="#">31.4.1</a>
	WriteCssStylesheet	Once	Never	<a href="#">31.4.1</a>
	WriteCssLink	Yes	No	<a href="#">31.4.1</a>
	XrefFormatIsXrefClass	Yes	No	<a href="#">31.7.7</a>
[HtmlOptions]	AlignAttributes	No	Yes	<a href="#">30.4</a>
	Basefont	No	Yes	
	NoFonts	Yes	No	

## 31.6 Linking to alternate CSS files

*In this section:*

§31.6.1 [Selecting a CSS file at run time](#) on page 597

§31.6.2 [Changing CSS files in the middle of a document](#) on page 598

§31.6.3 [Customizing the CSS link tag](#) on page 598

§31.6.4 [Using an alternate CSS link tag for Netscape 4](#) on page 599

### 31.6.1 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 **DITA2Go** 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

§37.1.1.2 [Understanding where you can define named macros](#) on page 680. You can modify the macro definition as needed; consult a JavaScript reference for syntax.

## 31.6.2 Changing CSS files in the middle of a document

To have **DITA2Go** reference different style sheets for output from different parts of your DITA document, you can use a macro to provide the CSS link, then insert PI markers in your document to signal a change of CSS file. To prevent **DITA2Go** 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
```

**DITA2Go** 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 PI marker in a paragraph *before* the split. You can use a **HTML Macro** PI marker, with content as follows:

```
<$$myAltCSS=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 file in your document, place in the project directory a file-specific configuration file that contains (only) the following setting:

```
[MacroVariables]
myAltCSS=SpecialCSS
```

See §42.1 [Using a different configuration for selected files](#) on page 765.

## 31.6.3 Customizing the CSS link tag

The generic CSS link tag **DITA2Go** 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 **DITA2Go** 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 §37.1 [Defining and invoking macros](#) on page 679):

```
[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 §37.3 [Using macro variables](#) on page 687) 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
```

### 31.6.4 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 `<b>...</b>` 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.

## 31.7 Assigning CSS classes

*In this section:*

- §31.7.1 [Understanding CSS class name restrictions](#) on page 600
- §31.7.2 [Mapping paragraph formats to CSS classes](#) on page 601
- §31.7.3 [Mapping character formats to tags or span classes](#) on page 602
- §31.7.4 [Assigning CSS classes to table formats](#) on page 603
- §31.7.5 [Assigning CSS classes to text and table footnotes](#) on page 603
- §31.7.6 [Assigning CSS classes based on Unicode character ranges](#) on page 603
- §31.7.7 [Using link format names as CSS class names](#) on page 604

[§31.7.8 Using CSS class names as tags for XML](#) on page 604

[§31.7.9 Omitting tags from CSS selectors](#) on page 605

[§31.7.10 Overriding CSS class for selected paragraphs](#) on page 605

See also:

[§30.11.2.6 Including or excluding the type list attribute](#) on page 588

## 31.7.1 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. **DITA2Go** imposes an internal limit of 128 characters on CSS class names.

To create class names from format names, **DITA2Go** does the following:

- removes or replaces spaces
- removes all non-alphanumeric characters
- replaces accented characters with their non-accented equivalents
- 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.

*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, **DITA2Go** 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: **DITA2Go** 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, **DITA2Go** changes all generated class names in output in lowercase.
- For standard HTML, HTML Help, and OmniHelp, class names generated from 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
```

## 31.7.2 Mapping paragraph formats to CSS classes

When you use CSS, by default **DITA2Go** maps each paragraph format name to a CSS class of the same name, applying to the name any needed transformations (see §31.7.1 [Understanding CSS class name restrictions](#) on page 600).

For a paragraph format, by default the class name in the **DITA2Go**-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 §30.2 [Mapping paragraph formats](#) on page 566), or else <p>. Unless you assign classes explicitly, the class name is based on the paragraph format name.

For example, suppose your document includes paragraph formats *Chap\_Title*, *Heading*, and *Body*, with the first two assigned HTML tags in [ParaTags]. **DITA2Go** would treat these formats as follows, provided `ClassIsTag=No` (see §31.7.8 [Using CSS class names as tags for XML](#) on page 604):

<u>FM format name</u>	<u>[ParaTags]</u>	<u>DITA2Go HTML output</u>	<u>DITA2Go CSS entry</u>
<i>Chap_Title</i>	Chap_Title=H1	<h1 class="chaptitle">	h1.chaptitle {...}
<i>SubHead</i>	SubHead=H2	<h2 class="subhead">	h2.subhead {...}
<i>Body</i>	(no setting)	<p class="body">	p.body {...}

**DITA2Go** 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 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, **DITA2Go** uses the [ParaTags] setting for backward compatibility. See §30.2.1 [Assigning HTML tags and attributes to paragraph formats](#) on page 566.

*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 §23.3.1 [Deriving XML tags from format and class names](#) on page 452.

### 31.7.3 Mapping character formats to tags or span classes

When you use CSS, **DITA2Go** generates any tags assigned to a character format in [CharTags]; see §30.3 [Mapping character formats](#) on page 569. By default, **DITA2Go** maps each 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 §31.7.1 [Understanding CSS class name restrictions](#) on page 600).

For example, suppose your document uses character format names *Emphasis*, *Prog Term*, and *Link*, with the first two assigned HTML tags in [CharTags]. **DITA2Go** would treat these formats as follows, provided `ClassIsTag=No` (see §31.7.8 [Using CSS class names as tags for XML](#) on page 604):

<u>FM format name</u>	<u>[CharTags]</u>	<u>DITA2Go HTML output</u>	<u>DITA2Go CSS entry</u>
<i>Emphasis</i>	Emphasis=em	<em>	em.emphasis {...}
<i>Prog Term</i>	Prog Term=code	<code>	code.progterm {...}
<i>Link</i>	(no setting)	<span class="link">	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 character format name not listed in [CharTags] is assigned to a span class of the same name as the format.

When `UseSpanAsDefault=No`, any character format name not listed in [CharTags] is skipped, and becomes just an override in HTML output.

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 `<span class="classname">` tags, to define character formats globally in CSS. For example, if you map character format *CodeBold* to `<span class="codebold">`, **DITA2Go** 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, **DITA2Go** uses the [CharTags] setting for backward compatibility. See §30.3 [Mapping character formats](#) on page 569.

Generic XML

For generic XML output, see §23.3.1 [Deriving XML tags from format and class names](#) on page 452.



### 31.7.4 Assigning CSS classes to table formats

To explicitly map individual 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
```

When you assign a CSS class name to a table format, if you are using output formats (see §7.7 [Configuring table output formats](#) on page 129), **DITA2Go** looks in the table configuration chain for a table format named *classname*.

See also:

§33.4.3 [Assigning a CSS class to a table](#) on page 633

### 31.7.5 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:

§30.10 [Converting footnotes to HTML or XML](#) on page 581

### 31.7.6 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.

**DITA2Go** 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 = -
```

### 31.7.7 Using link format names as CSS class names

To automatically use 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 §31.5 [Understanding how CSS affects other options](#) on page 596.

For DITA XML, the default value of XrefFormatIsXrefClass is Yes; see §24.3 [Specifying general options for DITA](#) on page 458.

### 31.7.8 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 §31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593.

When `ClassIsTag=No`, HTML tags and class names are assigned as described in §31.7.2 [Mapping paragraph formats to CSS classes](#) on page 601 and §31.7.3 [Mapping character formats to tags or span classes](#) on page 602.

For example, suppose your document includes paragraph formats *Chap\_Title*, *SubHead*, *Fig*, and *Body*, with the first two assigned HTML tags and the third assigned a class in `[ParaTags]`. **DITA2Go** 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>

### 31.7.9 Omitting tags from CSS selectors

By default, for HTML output **DITA2Go** writes CSS selectors as class names prefixed with the element tag.

To have **DITA2Go** 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.

### 31.7.10 Overriding CSS class for selected paragraphs

Paragraphs that have distinct purposes in your document should have distinct 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** PI 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** PI marker in the paragraph that *precedes* each such paragraph. The content of the PI marker would look like this:

```
<$$h2class="A">
```

To assign a non-default class to the very first paragraph in a file, you would have to create a chapter-specific configuration file, `filename.ini`, for that file, with content (for example):

```
[MacroVariables]
h2class=A
```

See §42.1.1 [Providing configuration files for individual ditamaps](#) on page 765.

See also:

§37 [Working with macros](#) on page 679

§38 [Working with processing instructions](#) on page 717

§42 [Overriding configuration settings](#) on page 765

## 31.8 Customizing CSS properties

In this section:

§31.8.1 [Specifying CSS <body> tag properties](#) on page 606

§31.8.2 [Specifying CSS size values and units of measurement](#) on page 607

§31.8.3 [Overriding styles in DITA2Go-generated CSS files](#) on page 608

§31.8.4 [Adjusting leading \(line spacing\) in CSS](#) on page 609

§31.8.5 [Preventing <font> tags from overriding CSS properties](#) on page 609

### 31.8.1 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 **DITA2Go**-generated CSS file. Or, you can direct **DITA2Go** 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 **DITA2Go** from automatically including a style entry for the `<body>` tag in a **DITA2Go**-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:*

§31.8.2 [Specifying CSS size values and units of measurement](#) on page 607

§31.8.3 [Overriding styles in DITA2Go-generated CSS files](#) on page 608

## 31.8.2 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 **DITA2Go**-generated CSS entries. You can direct **DITA2Go** 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 §31.8.1 [Specifying CSS <body> tag properties](#) on page 606.

You can specify how many decimal places **DITA2Go** 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:

[§31.8.1 Specifying CSS <body> tag properties](#) on page 606

[§31.8.3 Overriding styles in DITA2Go-generated CSS files](#) on page 608

### 31.8.3 Overriding styles in DITA2Go-generated CSS files

When you direct **DITA2Go** to generate a CSS file anew each time (that is, when [CSS]WriteCssStylesheet=Always), styles are updated from the formats in your 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:

[§31.8.1 Specifying CSS <body> tag properties](#) on page 606

[§31.8.2 Specifying CSS size values and units of measurement](#) on page 607

**Override CSS code** To override the style specification in a **DITA2Go**-generated CSS file for a particular 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, **DITA2Go** 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 **DITA2Go** does not use; or add CSS code for positioning, or to set anchor properties, or <body> properties, or special properties for nested items.

### 31.8.4 Adjusting leading (line spacing) in CSS

By default, **DITA2Go** 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 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`, but renders `margin top`, equivalent to Space Above, reasonably well.

### 31.8.5 Preventing <font> tags from overriding CSS properties

To keep <font> 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 §31.4.1 [Specifying CSS options in a DITA2Go configuration file](#) on page 593.





# 32 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:

- §32.1 [Locating graphics files for HTML](#) on page 611
- §32.2 [Specifying options for HTML graphics](#) on page 612
- §32.3 [Omitting graphics from HTML output](#) on page 613
- §32.4 [Selecting and modifying graphics](#) on page 613
- §32.5 [Positioning graphics in HTML output](#) on page 617
- §32.6 [Specifying HTML image attributes](#) on page 619
- §32.7 [Providing \(or omitting\) alternate text for images](#) on page 620
- §32.8 [Scaling images for HTML](#) on page 620
- §32.9 [Creating image maps for HTML](#) on page 622
- §32.10 [Supplying a background image or watermark](#) on page 624

See also:

- §40 [Working with graphics](#) on page 745

## 32.1 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 `<img>` tags do not need a path component, and whatever path information is already present in DITA 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`, **DITA2Go** omits any path information from references in generated `<img>` 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 **DITA2Go** can include the path in the generated `<img>` 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 §44.6 [Assembling files for distribution](#) on page 792); 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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

If you do not specify a value for GraphPath, the value of StripGraphPath determines whether **DITA2Go** includes the original path from your DITA document, or no path at all, in generated <img> 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 DITA document. This option sets the src attribute of the <img> 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 **DITA2Go** copy them for you. See §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796.

See also:

- §44.7 [Placing graphics files for distribution](#) on page 796
- §18.3.9 [Locating graphics files for HTML Help](#) on page 318
- §19.3.8 [Getting OmniHelp supporting files in the right place](#) on page 360
- §20.3.6.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 391
- §40.2.1.1 [Specifying graphics location for HTML](#) on page 747

## 32.2 Specifying options for HTML graphics

If some referenced graphics are already in a format appropriate for Web use, such as JPEG, GIF, or PNG (see §40.1.4 [Graphics formats for HTML](#) on page 746) you do not have to convert them.

Copy the graphics files into the project directory with the generated .htm files, or allow **DITA2Go** to copy them for you to the wrap directory; see §32.1 [Locating graphics files for HTML](#) on page 611. Unless you explicitly remap a name in the [GraphFiles] section, 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 DITA files that reference them, but they will be in the same directory as the generated HTML files, set the following option also (see §32.1 [Locating graphics files for HTML](#) on page 611):

```
[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 §40.2.1.2 [Substituting graphics files for HTML](#) on page 747):

```
[Graphics]
GraphSuffix=jpg
```

Use this setting when you convert referenced graphics with a third-party program; see §4.4 [Processing graphics](#) on page 77.

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
```

3. Specify file names with extensions for any individual exceptions (see §40.2.1.2 [Substituting graphics files for HTML](#) on page 747):

```
[GraphFiles]
newlogo.jpg=newlogo.gif
```

## 32.3 Omitting graphics from HTML 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 setting to eliminate any anchor paragraphs for the graphics:

```
[Graphics]
GraphWrapPara = No
```

*See also:*

§32.4.2 [Replacing or surrounding a graphic with macro code](#) on page 615

§32.4.3 [Omitting paragraph tags around graphics](#) on page 616

## 32.4 Selecting and modifying graphics

*In this section:*

§32.4.1 [Assigning properties to sets of graphics](#) on page 613

§32.4.2 [Replacing or surrounding a graphic with macro code](#) on page 615

§32.4.3 [Omitting paragraph tags around graphics](#) on page 616

### 32.4.1 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. The key in all these settings is the *GraphicID*, which is the base name of an image file, without path or extension.

*In this section:*

§32.4.1.1 [Using wildcards to assign properties to graphics](#) on page 614

[§32.4.1.2 Using PI markers to assign properties to graphics](#) on page 614

[§32.4.1.3 Specifying an image class for a graphic](#) on page 615

[§32.4.1.4 Creating named groups of graphics](#) on page 615

### 32.4.1.1 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:

[§3.6 Using wildcards in configuration settings](#) on page 65.

### 32.4.1.2 Using PI markers to assign properties to graphics

You can use PI markers in your DITA document to assign a property to a single graphic or to only a few graphics, or to exclude a graphic from a general assignment. Use either of these PI marker types:

**HTMConfig** Content is `[GraphSection]=Value`

**HTML Macro** Content is any HTML code

*HTMConfig for individual graphics*

Insert the **HTMConfig** PI 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** PI marker just before the `<image>` tag, and specify the scale factor as the PI marker content:

```
HTMConfig="[GraphScale]=75"
```

See [§42.2.9.4 Overriding graphic properties for HTML](#) on page 774.

*HTML Macro for a series of graphics*

You can use PI 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** PI 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** PI marker with content:

```
<$$scalepct=75>
```

Just after the graphics to be scaled, insert another **HTML Macro** PI marker with content:

```
<$$scalepct=100>
```

See §37.3 [Using macro variables](#) on page 687.

### 32.4.1.3 Specifying an image class for a graphic

To assign a CSS class to the `<img>` tag created for a graphic insert a **GraphClass** PI marker in text preceding the graphic. Make the content of the marker the name of the image class. See §38.3 [Adding attributes with PI markers](#) on page 721.

See also:

§32.6 [Specifying HTML image attributes](#) on page 619

### 32.4.1.4 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 (base file names) 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 an individual graphic file. 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** PI marker in text just before the graphic in your DITA document, with content as follows:

```
HTMConfig="[GraphGroup]=graphicgroupname"
```

## 32.4.2 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, **DITA2Go** 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 §32.4.3 <a href="#">Omitting paragraph tags around graphics</a> on page 616.								
<i>List exceptions by graphic ID</i>	<p>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:</p> <pre>[GraphReplaceMacros] aa12345= aa23456= *=&lt;\$YourMacro&gt;</pre> <p>See §32.4.1.1 <a href="#">Using wildcards to assign properties to graphics</a> on page 614.</p>								
<i>Use predefined macro variables</i>	<p>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:</p> <table> <tr> <td>&lt;\$\$_graphbase&gt;</td><td>File name for &lt;img src= /&gt; attribute, without extension</td></tr> <tr> <td>&lt;\$\$_graphsrc&gt;</td><td>File name for &lt;img src= /&gt; attribute, with extension</td></tr> <tr> <td>&lt;\$\$_graphorighigh&gt;</td><td>Original image height in pixels, before any [GraphScale], [GraphHigh], or [GraphWide] setting is applied</td></tr> <tr> <td>&lt;\$\$_graphorigwide&gt;</td><td>Original image width in pixels, before any [GraphScale], [GraphHigh], or [GraphWide] setting is applied</td></tr> </table> <p>If you assign code instead of a macro name, the code must be all on the same line.</p>	<\$\$_graphbase>	File name for <img src= /> attribute, without extension	<\$\$_graphsrc>	File name for <img src= /> 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
<\$\$_graphbase>	File name for <img src= /> attribute, without extension								
<\$\$_graphsrc>	File name for <img src= /> 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								
<i>Replace graphics with thumbnails</i>	<p>To show each graphic in a smaller size (a “thumbnail”), for example, you could specify something like the following (see §3.6 <a href="#">Using wildcards in configuration settings</a> on page 65):</p> <pre>[GraphReplaceMacros] *=&lt;\$Thumbnail&gt;  [Thumbnail] &lt;a href="&lt;\$\$_graphsrc&gt;"&gt;&lt;img src="&lt;\$\$_graphsrc&gt;" width="25%" /&gt;&lt;/a&gt;</pre> <p>For a way to provide thumbnails in the form of links to the graphics they replace, see §27.8.3.2 <a href="#">Using thumbnails for links to illustrations in HTML</a> on page 540.</p>								
<i>View full-size graphics on demand</i>	<p>If you scale down high-resolution bitmap images, 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:</p> <pre>[GraphReplaceMacros] aa4de33f=&lt;\$FullView&gt;  [FullView] &lt;a href="&lt;\$\$_graphsrc&gt;" target="_blank"&gt;&lt;img src="&lt;\$\$_graphsrc&gt;" width="&lt;\$\$_graphorigwide&gt;" height="&lt;\$\$_graphorighigh&gt;" /&gt;&lt;/a&gt;</pre>								

### 32.4.3 Omitting paragraph tags around graphics

By default, **DITA2Go** wraps each non-inline graphic in paragraph tags. If you are replacing graphics with macro code (see §32.4.2 [Replacing or surrounding a graphic with macro code](#) on page 615) 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
```

## 32.5 Positioning graphics in HTML output

HTML positioning attributes are not compatible with CSS; therefore, by default, **DITA2Go** omits these attributes for HTML output. Use the positioning properties available in format configuration files instead; see §7 [Configuring output formats](#) on page 109.

*In this section:*

- §32.5.1 [Aligning anchored graphics](#) on page 617
- §32.5.2 [Indenting images \(deprecated\)](#) on page 617
- §32.5.3 [Adding space above an image](#) on page 619
- §32.5.4 [Eliminating space above or below graphics in table cells](#) on page 619

*See also:*

- §32.4.3 [Omitting paragraph tags around graphics](#) on page 616

### 32.5.1 Aligning anchored graphics

When you use CSS, you can center graphics with **DITA2Go** macros. For example, to center all images:

```
[GraphStartMacros]
* = <div class="img">

[GraphEndMacros]
* = </div>
```

See §32.4.2 [Replacing or surrounding a graphic with macro code](#) on page 615.

Include in CSS:

```
div.img {text-align: center; }
```

Or, if **DITA2Go** maintains CSS for your project (see §31.8.3 [Overriding styles in DITA2Go-generated CSS files](#) on page 608):

```
[CSSEndMacro]
div.img {text-align: center; }
```

You have to use `<div>` because CSS applies `text-align` only to block elements, and `<img>` is not a block element.

### 32.5.2 Indenting images (*deprecated*)

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.

**Note:** This technique has been superseded by format configuration properties, which use CSS; see §7 [Configuring output formats](#) on page 109.

**DITA2Go** can indent all non-inline graphics in HTML output using a technique invented by Chuck Musciano

<http://www.drdoobs.com/184411862>

This technique consists of placing, at the start of the line that contains the graphic, a one-pixel transparent GIF image, `1p.gif`, with a `width` attribute that produces the required

indent in pixels. See §37.2.2 [Modifying DITA2Go-supplied macro definitions](#) on page 685.

To use the built-in **DITA2Go** 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, **DITA2Go** 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`), **DITA2Go** writes the spacer image file to the specified path. Then **DITA2Go** generates references of the form:

```

```

This can be important if you place graphics anywhere but the project directory (see §32.1 [Locating graphics files for HTML](#) on page 611).

*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 §32.4.1 [Assigning properties to sets of graphics](#) on page 613.

*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 §3.6 [Using wildcards in configuration settings](#) on page 65.

### 32.5.3 Adding space above an image

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 §32.4.1 [Assigning properties to sets of graphics](#) on page 613.

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.

See also:

§32.8.1 [Excluding image size attributes from HTML](#) on page 620

### 32.5.4 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 §30.2.4 [Stripping paragraph properties](#) on page 568.

## 32.6 Specifying HTML image attributes

You can specify attributes for the `<img>` tag in either of the following ways:

[Configuration-file settings](#)

[PI markers](#).

*Configuration-file settings*

To specify `<img>` 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 borders from all graphics:

```
[GraphAttr]
*= border="0"
```

*PI markers*

You can define a PI marker that has a name beginning with **Graph** and ending with the name of a valid `<img>` tag attribute (see §38.3 [Adding attributes with PI markers](#) on page 721). The content of the marker becomes the value of the attribute for the next image in your document. The PI marker overrides any configuration-file setting for the same attribute for that image. See §42.2 [Overriding settings with PI markers or macros](#) on page 766.

See also:

§32.8 [Scaling images for HTML](#) on page 620

§34.2 [Applying WAI markup to images](#) on page 650

## 32.7 Providing (or omitting) alternate text for images

Most current browsers display the content of the `alt` attribute of an `<img>` 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, **DITA2Go** 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 `<img>` tag `alt` attribute in any of the following ways:

[Include the text in a paragraph](#)

[Insert the text with a PI marker](#)

[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 DITA document, just before the `<image>` tag, 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 §34.2.2 [Assigning WAI image attributes with dedicated formats](#) on page 651.

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 PI marker*

To provide alternate text with a PI marker, insert a PI marker of type **GraphAlt** in your DITA document, just before the `<image>` tag. The content of the **GraphAlt** PI marker becomes the value of the `alt` attribute for the next image. See §34.2.3 [Assigning WAI image attributes with PI markers](#) on page 651 and §38.3 [Adding attributes with PI markers](#) on page 721.

*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
```

## 32.8 Scaling images for HTML

*In this section:*

§32.8.1 [Excluding image size attributes from HTML](#) on page 620

§32.8.2 [Adjusting image size for selected graphics](#) on page 621

§32.8.3 [Specifying px units for graphics sized in pixels](#) on page 621

### 32.8.1 Excluding image size attributes from HTML

By default, **DITA2Go** 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:

§23.3.2 [Eliminating HTML attributes and tags for generic XML](#) on page 452

§24.7.3 [Omitting size attributes from images for DITA output](#) on page 484

§26.7.3 [Omitting size attributes from images for DocBook](#) on page 521

## 32.8.2 Adjusting image size for selected graphics

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
```

*Suppress scaling* To suppress scaling for selected graphics:

```
[GraphScale]
GraphID = 0
```

Setting the percent to zero suppresses scaling because **DITA2Go** 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.

## 32.8.3 Specifying px units for graphics sized in pixels

By default, for all HTML and XML outputs except JavaHelp, **DITA2Go** 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 **DITA2Go** 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.

## 32.9 Creating image maps for HTML

*In this section:*

§32.9.1 [Providing alternate text for a hotspot in an image map](#) on page 622

§32.9.2 [Specifying jumps from image maps in framesets](#) on page 623

### 32.9.1 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 `<img>` tag, you cannot specify alternate text for the `<area>` tag with a PI 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:*

§32.9.1.1 [Assigning alternate text to an image-map hotspot](#) on page 622

§32.9.1.2 [Using the title attribute for alternate text for a hotspot](#) on page 623

#### 32.9.1.1 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 **DITA2Go** to use the `title` attribute instead; see §32.9.1.2 [Using the title attribute for alternate text for a hotspot](#) on page 623.

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>
<a href="#">Link via HyperLink:</a>	Base name of the destination HTML file, if the hotspot link is a <b>HyperLink</b> PI marker
<a href="#">Link via HyperJump:</a>	<b>HyperAnchor</b> PI marker content, if the hotspot link is a <b>HyperJump</b> PI marker
<a href="#">Multiple links:</a>	GraphicID followed by # followed by destination ID (one of the other two), to distinguish among multiple links to the same destination

*Link via  
HyperLink*

Suppose you use a **HyperLink** PI marker for an image-map link, with the following marker content:

```
<?dhtml HyperLink="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 **HyperLink** PI marker content looks like this:

```
<?dthtm HyperLink="http://www.chezmoi.com/mylife.htm#siblings" ?>
```

You would assign alternate text as follows:

```
[GraphALT
mylife#siblings = alternate text for hotspot
```

#### Link via HyperJump

Suppose your image-map link is a **HyperJump** PI marker, with a destination in the same document; for example:

```
<?dthtm HyperJump="awards.dita:firstplace" ?>
```

You would assign alternate text for the hotspot to destination identifier `firstplace`, which is the target **HyperAnchor** PI marker content:

```
[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
```

### 32.9.1.2 Using the title attribute for alternate text for a hotspot

When you provide alternate text for a hotspot in an image map, by default **DITA2Go** 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 **DITA2Go** 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`, **DITA2Go** 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 `<img>` tags (see §32.7 [Providing \(or omitting\) alternate text for images](#) on page 620), that text gets transferred to the `<img>` tag `title` attribute, and you lose the content of the `<img>` tag `alt` attribute.

`UseTitleForAlt` affects only alt content added for `<img>` tags and hotspot `<area>` tags in the `[GraphALT]` section, not alt content added for `<img>` tags via PI markers or via the `[HTMLParaStyles]` `Alt` property.

### 32.9.2 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 §22.13 [Using framesets](#) on page 443).
- 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
```

## 32.10 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 }
```

# 33 Converting tables to HTML

---

Before employing the configuration settings described in this section:

- To specify output formats and other options for DITA table types, see §6.9 [Specifying formats and options for tables](#) on page 103.
- To map DITA table elements to output formats, see §6.3.2 [Mapping table outputclass attributes to formats](#) on page 90.
- To define properties for each table format, see §7.7 [Configuring table output formats](#) on page 129.

You might find that you can achieve exactly the appearance you want for tables in HTML without any of the settings described here.

Topics include:

- §33.1 [Assigning properties to tables](#) on page 625
- §33.2 [Defining sets of tables](#) on page 626
- §33.3 [Specifying table structure](#) on page 627
- §33.4 [Specifying table attributes](#) on page 632
- §33.5 [Positioning tables, table titles, and table footnotes](#) on page 640
- §33.6 [Using macros to control table properties](#) on page 642
- §33.7 [Converting tables to paragraphs](#) on page 647

**DITA2Go** supports WAI (Web Accessibility Initiative) markup for HTML tables; see:

- §34 [Generating WAI markup for HTML](#) on page 649
- §35 [Identifying HTML table structure for WAI](#) on page 657
- §36 [Marking HTML table cells for WAI](#) on page 667

## 33.1 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 **DITA2Go**.

*In this section:*

- §33.1.1 [Understanding precedence of assignment methods](#) on page 625
- §33.1.2 [Overriding default table and cell properties and attributes](#) on page 626

### 33.1.1 Understanding precedence of assignment methods

Many settings for selected tables (and for table rows and cells) can be specified several ways. [Table 33-1](#) 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 33-1 Precedence of table and cell property assignment methods**

Precedence	Type	Assignment method	Ref.
Highest	PI marker	Content of <b>Config</b> or <b>HTMConfig</b> PI marker, or PI marker whose name starts with <b>Table</b> , <b>Row</b> , or <b>Cell</b> and ends with an attribute name	<a href="#">33.4.4</a>
	DITA2Go macro	Code in [Table*Attributes] or [Table*Macros]	<a href="#">33.6</a>
	Configuration setting	Settings in [TableAccess] or [TableAttributes]	<a href="#">33.3.2.6</a> , <a href="#">33.4.1</a>
	Configuration setting	Settings in [Attributes]	<a href="#">33.4.1</a>
Lowest	Configuration setting	Settings in [Tables]	<a href="#">33.3.2</a> , <a href="#">33.4.8</a>

### 33.1.2 Overriding default table and cell properties and attributes

To override table structure properties, use settings in the [TableAccess] section; see §33.3.2.6 [Overriding row and column group settings](#) on page 630.

To override table display attributes, use settings in the [TableAttributes] section; see §33.4.2 [Overriding attributes for selected tables](#) on page 633.

To fine-tune properties for selected tables or cells, use PI markers or macros; see §33.4.4 [Using PI markers to assign attributes to tables, rows, or cells](#) on page 634 and §33.6 [Using macros to control table properties](#) on page 642.

## 33.2 Defining sets of tables

You can set table-specific properties, and specify overrides to table defaults, according to TableID (not recommended) or table group; or you can use wildcards to make any table-specific setting apply to all or a subset of tables:

- TableID:* The value of the `id` attribute you have specified for the table element in DITA.
- Table group:* A group of tables you define, using configuration PI markers or TableIDs to specify group membership. See §33.2.1 [Creating table groups](#) on page 626.
- Wildcard set:* An informal set of tables identified with wildcards. See §33.2.2 [Using wildcards to specify table sets](#) on page 627.

### 33.2.1 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 PI markers.](#)

*Create table groups in the configuration file*

To create table groups and assign tables to groups in the configuration file:

```
[TableGroup]
; TableID = group name used in other Table sections for ID
; the filter first looks for TableID, then group
TableID = tablegroupname
```

Create table  
groups with  
configuration PI  
markers

Another way to create a table group, or assign tables to existing table groups, is to insert configuration PI markers in the tables in your DITA document, with content as follows:

```
[TableGroup]=tablegroupname
```

### 33.2.2 Using wildcards to specify table sets

You can specify an informal group of tables by using wildcards with partial TableID or table-group names; see §3.6 [Using wildcards in configuration settings](#) on page 65.

**DITA2Go** 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 in groups that start with ac
- single spacing after all other tables.

## 33.3 Specifying table structure

If the tables in your document are complex, you might need to specify which cells belong to headers or footers. Also, you might want to specify whether **DITA2Go** 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 §33.3.2.6 [Overriding row and column group settings](#) on page 630 and §33.3.3.2 [Specifying different header and footer counts for selected tables](#) on page 631.

*In this section:*

§33.3.1 [Choosing the table structure model](#) on page 627

§33.3.2 [Identifying row and column groups and header cells](#) on page 628

§33.3.3 [Identifying table headers and footers](#) on page 630

### 33.3.1 Choosing the table structure model

By default, **DITA2Go** 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 UseCALSMoDel=Yes, **DITA2Go** uses the CALS table model to convert tables. This is the default for generic XML, DocBook XML, and DITA XML.

When UseCALSMoDel=No, **DITA2Go** uses the HTML table model to convert tables. This is the default for HTML and XHTML.

## 33.3.2 Identifying row and column groups and header cells

*In this section:*

§33.3.2.1 [Using browser-dependent HTML tags for tables](#) on page 628

§33.3.2.2 [Designating table header cells](#) on page 628

§33.3.2.3 [Enumerating table column groups](#) on page 628

§33.3.2.4 [Wrapping table row groups](#) on page 629

§33.3.2.5 [Positioning table footer rows \(deprecated\)](#) on page 629

§33.3.2.6 [Overriding row and column group settings](#) on page 630

### 33.3.2.1 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, **DITA2Go** 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 **DITA2Go** to use these HTML table tags. [Table 33-2](#) shows the settings available. These settings apply to all tables in your document. To override a setting for one or more tables, see §33.3.2.6 [Overriding row and column group settings](#) on page 630.

**Table 33-2 Browser-dependent HTML tags for tables**

[Tables] setting	Default value	Purpose
UseTbHeaderCode	No	Use <code>&lt;th&gt;</code> for header cells; default is to use <code>&lt;td&gt;</code> for all cells
ColGroupElements	No	List <code>&lt;colgroup&gt;</code> elements before first table row; enables <code>scope="colgroup"</code> , each ColGroup head starts a new <code>&lt;colgroup&gt;</code>
HeadFootBodyTags	No	Wrap table rows in <code>&lt;thead&gt;</code> , <code>&lt;tfoot&gt;</code> , and <code>&lt;tbody&gt;</code> groups; enables <code>scope="rowgroup"</code> , each RowGroup head starts a new <code>&lt;tbody&gt;</code>

### 33.3.2.2 Designating table header cells

The default for HTML tables generated by **DITA2Go** is not to use the `<th>` tag to distinguish header cells from body cells. However, you can direct **DITA2Go** to identify header cells:

```
[Tables]
; UseTbHeaderCode = No (default, always use <td...>)
; or Yes (use <th...>)
UseTbHeaderCode=No
```

If you specify `UseTbHeaderCode=Yes`, **DITA2Go** generates `<th>` elements for all header cells in your tables.

### 33.3.2.3 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 §35 [Identifying HTML table structure for WAI](#) on page 657 for more information. However, you can use this setting also to add CSS `class` attributes.

**DITA2Go** 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 §31 [Setting up CSS for HTML](#) on page 591). To use `<col>` elements, specify them in `[TableStartMacros]` (see §33.6.1 [Invoking macros around tables](#) on page 642), 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 §33.3.2.6 [Overriding row and column group settings](#) on page 630.

### 33.3.2.4 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, **DITA2Go** wraps table rows in groups.

To prevent **DITA2Go** 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`, **DITA2Go** wraps table rows with `<thead>`, `<tbody>`, and `<tfoot>` tags, as follows:

- All rows that are included in the table header are wrapped in `<thead>...</thead>`.
- All rows that are included in the table footer 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 §35 [Identifying HTML table structure for WAI](#) on page 657 for more information. However, you can also use this setting also to add CSS `class` attributes.

### 33.3.2.5 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 **DITA2Go** 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 §33.3.2.4 [Wrapping table row groups](#) on page 629), 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.

### 33.3.2.6 Overriding row and column group settings

You can override the default value of `HeadFootBodyTags` or `ColGroupElements` for table groups:

```
[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
```

### 33.3.3 Identifying table headers and footers

To designate footer rows (a feature not present in DITA) for selected tables, see §6.9 [Specifying formats and options for tables](#) on page 103.

*In this section:*

§33.3.3.1 [Specifying default header and footer counts for all tables](#) on page 631



§33.3.3.2 [Specifying different header and footer counts for selected tables](#) on page 631

### 33.3.3.1 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 §35 [Identifying HTML table structure for WAI](#) on page 657 for more information. [Table 33-3](#) shows the settings.

**Table 33-3** 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,
TableFooterRows	0	FRowsN	Number of rows (counting from the bottom) to treat as footer rows for row-grouping purposes, 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 DITA-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 rows are misclassified as body rows in DITA. If you consistently use DITA header (there is no DITA concept of footer) rows for the headers in your tables, and you provide an `@outputclass` for rows to be treated as footer rows, you do not need either of these settings. See §6.9 [Specifying formats and options for tables](#) on page 103.

### 33.3.3.2 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:

HColsN	Treats cells in the first <i>N</i> columns (counting from the left) as row headers; tags the cells <th> if [Tables]UseTbHeaderCode=Yes. See §33.3.2.2 <a href="#">Designating table header cells</a> on page 628).
HRowsN	Treats cells in the first <i>N</i> rows (counting from the top) as column headers; tags the cells <th> if [Tables]UseTbHeaderCode=Yes. See §33.3.2.3 <a href="#">Enumerating table column groups</a> on page 628).
FRowsN	Treats the last <i>N</i> rows (counting from the bottom) as footer rows; wraps them in a <tfoot>...</tfoot> element if [Tables]HeadFootBodyTags=Yes. See §33.3.2.4 <a href="#">Wrapping table row groups</a> on page 629).

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 **DITA2Go** settings for header cells work just as though the cells were tagged <th>.

## 33.4 Specifying table attributes

You can exclude auto-generated attributes from HTML or XML output, and have **DITA2Go** include only those for which you specify explicit values.

*In this section:*

- §33.4.1 [Specifying attributes for all tables](#) on page 632
- §33.4.2 [Overriding attributes for selected tables](#) on page 633
- §33.4.3 [Assigning a CSS class to a table](#) on page 633
- §33.4.4 [Using PI markers to assign attributes to tables, rows, or cells](#) on page 634
- §33.4.5 [Specifying attributes for table rows](#) on page 634
- §33.4.6 [Specifying attributes for table cells](#) on page 635
- §33.4.8 [Adjusting borders, cell spacing, and cell padding](#) on page 636
- §33.4.9 [Determining the width of table columns](#) on page 638
- §33.4.10 [Deciding what to do with empty paragraphs in table cells](#) on page 640

*See also:*

- §33.1.1 [Understanding precedence of assignment methods](#) on page 625
- §33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644

### 33.4.1 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 §33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644.

*No border,  
cellspacing, or  
cellpadding*

If you list attributes for the <table> tag, do *not* include border, cellspacing, or cellpadding; if you do, **DITA2Go** writes duplicate assignments for any of these you specify in [Attributes]. Use one of the following instead:

- Settings in the [Tables] section; see §33.4.8 [Adjusting borders, cell spacing, and cell padding](#) on page 636.
- Attributes in the [TableAttributes] section; see §33.4.2 [Overriding attributes for selected tables](#) on page 633.

Also see §33.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 637.

### 33.4.2 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 or a table group name, and you can use wildcards in the name. See §33.2.1 [Creating table groups](#) on page 626.

On the right of the = sign you can include any arbitrary HTML, even macros (see §37 [Working with macros](#) on page 679) and JavaScript (perhaps something like onmouseover="javascript:doSomething(now)"). Just keep it all on the same line.

For example, for cell borders, 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 §33.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 637 for special constraints on specifying values for border, cellspacing, and cellpadding.

*See also:*

§33.1.1 [Understanding precedence of assignment methods](#) on page 625

### 33.4.3 Assigning a CSS class to a table

The default CSS class for a table is the table @outputclass; failing that, the 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, and does not connect to format definitions in a table-format configuration file.

See also:

§7.7 [Configuring table output formats](#) on page 129

§31.7.4 [Assigning CSS classes to table formats](#) on page 603

### 33.4.4 Using PI markers to assign attributes to tables, rows, or cells

If you give a PI marker type a name that starts with **Table**, **Row**, or **Cell**, DITA2Go uses the content of the marker for the value of whatever HTML attribute is designated by the rest of the PI 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 PI marker takes precedence over values of the same attribute assigned any other way; see [Table 33-1](#) on page 626.

For example, to guarantee that a certain table cell is top-aligned in HTML regardless of its properties in DITA or any properties assigned in the configuration file, you could insert a PI marker of type **CellValign** just before the cell in DITA, and make the content of that marker `top`. In HTML, the resulting tag for that cell would be `<td valign="top">`.

See also:

§34.1.3 [Inserting PI markers for WAI attributes](#) on page 650

§38.3 [Adding attributes with PI markers](#) on page 721

### 33.4.5 Specifying attributes for table rows

You can specify attributes for the `<tr>` element with any of the following:

[Paragraph format](#)

[Table format](#)

[Attribute PI marker.](#)

See also:

§33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644.

*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 §42.2.9.1 [Overriding paragraph and character format properties](#) on page 771.

*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 §33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644.

You can use a macro for the assignment in `[TableRowAttributes]`.

*Attribute PI  
marker*

To assign an attribute to an individual row, place a PI 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 PI marker of type **RowClass** inside any cell in the row. See §38.3 [Adding attributes with PI markers](#) on page 721.

### 33.4.6 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 PI marker.](#)

Also see §33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644.

*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 §35.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 662. 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 §33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644. You can use a macro for the assignment in `[TableCellAttributes]`.

*Attribute PI  
marker*

To specify attributes for an individual cell, place a PI 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 PI marker of type **CellClass** inside the cell, and make the marker content the name of the class. See §38.3 [Adding attributes with PI markers](#) on page 721.

### 33.4.7 Eliminating automatically generated attributes

**DITA2Go** automatically generates the following attributes for tables in HTML (but not XML):

[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 §33.4.8.3 [Excluding border, cellspacing, and cellpadding attributes](#) on page 638.

*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 **DITA2Go** excludes these automatically generated attributes; however, **DITA2Go** still includes any of these attributes that you specify explicitly in markers or in either of the following sections:

- [Attributes]; see §33.4.1 [Specifying attributes for all tables](#) on page 632)
- [TableAttributes]; see §33.4.2 [Overriding attributes for selected tables](#) on page 633.

See §23.1 [Setting up a generic XML project](#) on page 449 and §23.3.2 [Eliminating HTML attributes and tags for generic XML](#) on page 452.

### 33.4.8 Adjusting borders, cell spacing, and cell padding

Unless you change their values in a configuration file, 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 this section:*

§33.4.8.1 [Specifying default border, cellspacing, and cellpadding values](#) on page 636

§33.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 637

§33.4.8.3 [Excluding border, cellspacing, and cellpadding attributes](#) on page 638

#### 33.4.8.1 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:

§33.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 637

§33.4.8.3 [Excluding border, cellspacing, and cellpadding attributes](#) on page 638

### 33.4.8.2 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, **DITA2Go** 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 **DITA2Go** 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, **DITA2Go** 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">
```



### 33.4.8.3 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 PI 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, **DITA2Go** includes any values you specify for these attributes in PI markers or in the following sections:

- `[Attributes]`; see §33.4.1 [Specifying attributes for all tables](#) on page 632
- `[TableAttributes]`; see §33.4.2 [Overriding attributes for selected tables](#) on page 633.

*See also:*

- §33.4.8.1 [Specifying default border, cellspacing, and cellpadding values](#) on page 636
- §33.4.8.2 [Taming border, cellspacing, and cellpadding settings](#) on page 637

## 33.4.9 Determining the width of table columns

The DITA table element is based on the OASIS model, which does not really provide a way to specify overall table size; all you get is `@pgwide`, which for XHTML means nothing at all. The `colspec` element has `@colwidth`, which “describes the column width”. For the `simpletable` element, DITA provides only `@relcolwidth`, with its odd `n*` method of setting percentages. Your best bet is to use CSS syntax and semantics for column “width”.

By default, table columns are adaptively sized in HTML output. You can specify a different sizing method in the configuration file. You can also override the default sizing method for particular tables.

*In this section:*

- §33.4.9.1 [Specifying a method for determining table column widths](#) on page 638
- §33.4.9.2 [Overriding the default table column sizing method](#) on page 639
- §33.4.9.3 [Maintaining the width of table columns via relative sizing](#) on page 639

### 33.4.9.1 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](#) 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 table groups, see §33.4.9.2 [Overriding the default table column sizing method](#) on page 639.

<i>Adaptive table sizing</i>	When <code>TableSizing=Adaptive</code> (the default), <b>DITA2Go</b> 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 <code>Fixed</code> or <code>Percent</code> .
<i>Relative table sizing</i>	When <code>TableSizing=Percent</code> , <b>DITA2Go</b> computes the width of each column as a percent of the table width and gives each table cell a width attribute with a value expressed as a percent of the full table width. See §33.4.9.3 <a href="#">Maintaining the width of table columns via relative sizing</a> on page 639.
<i>Fixed table sizing</i>	When <code>TableSizing=Fixed</code> , <b>DITA2Go</b> 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.

### 33.4.9.2 Overriding the default table column sizing method

To override the default table sizing method for selected table groups:

```
[TableSizing]
; table ID = type of sizing: Adaptive (default), Fixed (pixels),
; Percent
TableID = Adaptive
```

This setting overrides the default method specified by `[Tables]TableSizing`; see §33.4.9.1 [Specifying a method for determining table column widths](#) on page 638.

### 33.4.9.3 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 DITA document, or if no widths are specified in DITA. Otherwise, use one of the settings described in §33.4.9.1 [Specifying a method for determining table column widths](#) on page 638, 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 §33.4.2 [Overriding attributes for selected tables](#) on page 633.

### 33.4.10 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, **DITA2Go** 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 **DITA2Go** do any of the following:

[Omit empty paragraph tags](#)

[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 DITA (contains only empty elements) in HTML output that cell would consist of only <td></td>.

When RemoveEmptyTableParagraphs=No, the tags for empty elements are retained in table cells.

*Provide content  
for empty  
paragraphs*

To specify text content for otherwise empty elements 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: &nbsp;.

*Retain empty  
paragraph tags*

To retain paragraph tags but omit text content for empty elements in table cells:

```
[Tables]
RemoveEmptyTableParagraphs = No
EmptyTbCellContent =
```

When RemoveEmptyTableParagraphs=Yes, EmptyTbCellContent has no effect.

## 33.5 Positioning tables, table titles, and table footnotes

*In this section:*

§33.5.1 [Indenting tables \(deprecated\)](#) on page 641

§33.5.2 [Configuring and positioning table titles](#) on page 641

§33.5.3 [Positioning table footnotes](#) on page 642

### 33.5.1 Indenting tables (*deprecated*)

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.

**Note:** This technique has been superseded by format configuration properties, which use CSS; see §7 [Configuring output formats](#) on page 109.

Because there is no indent attribute for tables in HTML, **DITA2Go** places a spacer graphic just before the table; see §32.5.2 [Indenting images \(deprecated\)](#) on page 617.

To use the spacer graphic:

```
[HTMLOptions]
; UseSpacers = No (default)
; or Yes, use to position tables and graphics
UseSpacers = Yes
```

See §32.5.2 [Indenting images \(deprecated\)](#) on page 617.

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` from being indented:

```
[TableIndents]
af123456=60
ag*=0
```

### 33.5.2 Configuring and positioning table titles

To specify use and placement of 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`.

### 33.5.3 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:

[§30.10.1 Configuring and placing footnotes](#) on page 582

[§31.7.5 Assigning CSS classes to text and table footnotes](#) on page 603.

## 33.6 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 **DITA2Go** macros (see [§37.9.4 Assigning macros to graphics or tables for HTML](#) on page 713). You can even use macros to wrap a table in another table, to get special effects.

In this section

[§33.6.1 Invoking macros around tables](#) on page 642

[§33.6.2 Adding space before tables](#) on page 643

[§33.6.3 Adjusting space after tables](#) on page 643

[§33.6.4 Turning processing on and off around selected tables](#) on page 643

[§33.6.5 Specifying row-group, row, and cell attributes with macros](#) on page 644

[§33.6.6 Capturing table row and column counts with variables](#) on page 645

[§33.6.7 Selectively modifying table text with macros: an example](#) on page 645

### 33.6.1 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.

### 33.6.2 Adding space before tables

To add space before all tables:

```
[TableBeforeMacros]
*=<br>
```

To add space before a specific table, use its TableID (see §33.2 [Defining sets of tables](#) on page 626); for example:

```
[TableBeforeMacros]
ae1001207=<br>
```

**DITA2Go** 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 §33.6.1 [Invoking macros around tables](#) on page 642.

### 33.6.3 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. **DITA2Go** 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 in groups that start with `ac`
- single spacing after all other tables.

See §33.6.1 [Invoking macros around tables](#) on page 642.

### 33.6.4 Turning processing on and off around selected tables

Suppose you use two-cell tables 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 §33.7.2 [Removing table-specific tags from selected tables](#) on page 647) 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 §32.4.1.4 [Creating named groups of graphics](#) on page 615), with group name *NoteIcons*:

```
[GraphReplaceMacros]
NoteIcons = <$$nothing=1>
```

See §32.3 [Omitting graphics from HTML output](#) on page 613.

### 33.6.5 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 table groups, any one of:

```
[TableAccess]
TableID = HFBTags
TableID = Scope
TableID = ScopeRowGroup
```

*See also:*

§33.3.2.4 [Wrapping table row groups](#) on page 629

§33.3.2.6 [Overriding row and column group settings](#) on page 630.

*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 §35.1 [Identifying table rows and columns](#) on page 657 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:

§33.4.5 [Specifying attributes for table rows](#) on page 634

**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>
```

### 33.6.6 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:

§33.6.1 [Invoking macros around tables](#) on page 642

§37.3.4 [Using predefined macro variables](#) on page 691

### 33.6.7 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 DITA the text is not differentiated with an inline element.
- 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:

[§33.2.2 Using wildcards to specify table sets](#) on page 627

[§33.6.1 Invoking macros around tables](#) on page 642

[§37.3 Using macro variables](#) on page 687

Assign coding  
options to table-  
cell formats

Turn off the HTML paragraph tag that **DITA2Go** 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:

[§30.2.4 Stripping paragraph properties](#) on page 568

[§37.3.2 Assigning values to macro variables](#) on page 688

[§37.9.3 Surrounding or replacing text with code or macros](#) on page 711

[§37.3.5.2 Inserting code with the CodeStore property](#) on page 693

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:

§37.6.4 [Using control structures in expressions](#) on page 704

§37.6.5 [Specifying substrings in expressions](#) on page 706

## 33.7 Converting tables to paragraphs

To use content stored in tables as ordinary non-table content, you can direct **DITA2Go** 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:

§33.7.1 [Removing table-specific tags from all tables](#) on page 647

§33.7.2 [Removing table-specific tags from selected tables](#) on page 647

§33.7.3 [Removing table-specific tags from complex tables](#) on page 648

See also:

§33.6.4 [Turning processing on and off around selected tables](#) on page 643

### 33.7.1 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`, **DITA2Go** 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, **DITA2Go** 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 §27.3.1 [Designating split points](#) on page 526 and §27.5.2.2 [Assigning a title with a paragraph format](#) on page 532.

### 33.7.2 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 PI 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 §33.6.1 [Invoking macros around tables](#) on page 642 and §33.6.4 [Turning processing on and off around selected tables](#) on page 643.

*Strip table tags  
with Config PI  
markers*

To remove table tags from individual tables, you can surround each table with **Config PI** markers:

<u>Config PI marker content</u>	<u>Marker placement</u>
[Tables]StripTable=1	Before the table anchor
[Tables]StripTable=0	After the table

See §42.2.8 [Overriding fixed-key configuration settings](#) on page 770.

### 33.7.3 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 §33.6 [Using macros to control table properties](#) on page 642. You can also use format-related macros; see §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

# 34 Generating WAI markup for HTML

---

**DITA2Go** supports WAI (Web Accessibility Initiative) guidelines for authoring HTML content intended to be accessible to people with disabilities. Topics include:

§34.1 [Comparing DITA2Go markup methods for WAI](#) on page 649

§34.2 [Applying WAI markup to images](#) on page 650

§34.3 [Applying WAI markup to links](#) on page 651

§34.4 [Applying WAI markup to tables](#) on page 652

*See also:*

§38 [Working with processing instructions](#) on page 717

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 **DITA2Go**, see:

<http://www.w3.org/WAI/eval/Overview.html>

## 34.1 Comparing DITA2Go markup methods for WAI

*In this section:*

§34.1.1 [Choosing a markup method for WAI attributes](#) on page 649

§34.1.2 [Using paragraph formats for WAI attributes](#) on page 649

§34.1.3 [Inserting PI markers for WAI attributes](#) on page 650

### 34.1.1 Choosing a markup method for WAI attributes

**DITA2Go** provides the following ways to specify WAI attributes for HTML:

- Assign a **DITA2Go** property that represents a WAI attribute to a 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 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 PI marker named after a WAI attribute, and make the marker text the attribute value.

Macros and macro variables can be referenced in the attributes, both from PI markers and from text identified as attribute content.

### 34.1.2 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.

*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 **DITA2Go** 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 §35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661.

*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; **DITA2Go** 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.

### 34.1.3 Inserting PI markers for WAI attributes

You can insert PI markers in (or just before) items that require WAI markup.

*Marker name is significant*

If you give a PI marker type a name that starts with **Table**, **Cell**, **Graph**, or **Link**, **DITA2Go** automatically makes the marker text the value of whatever HTML attribute is named by the rest of the PI marker-type name, and puts the attribute and its value in the generated `<table>`, `<th>`, `<td>`, `<img>`, or `<a>` tag.

## 34.2 Applying WAI markup to images

*In this section:*

§34.2.1 [Following WAI guidelines for images](#) on page 650

§34.2.2 [Assigning WAI image attributes with dedicated formats](#) on page 651

§34.2.3 [Assigning WAI image attributes with PI markers](#) on page 651

### 34.2.1 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>

**DITA2Go** 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 PI marker.

### 34.2.2 Assigning WAI image attributes with dedicated formats

You can designate a paragraph format whose content will be the text alternative for the next image. 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 `<img>` 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 `<img>` 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.

### 34.2.3 Assigning WAI image attributes with PI markers

You can use PI 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 PI marker. Each PI marker name must start with `Graph` and end with the name of the attribute. Valid PI marker names are as follows

<b>GraphAlt</b>	Short text equivalent of image; adds <code>alt</code> attribute.
<b>GraphLongdesc</b>	Path to file with image description; adds <code>longdesc</code> attribute.
<b>GraphTitle</b>	Long text description of image; adds <code>title</code> attribute.

For example, to provide an `alt` attribute for each graphic, place a **GraphAlt** PI marker just before the image.

## 34.3 Applying WAI markup to links

*In this section:*

§34.3.1 [Following WAI guidelines for links](#) on page 652

§34.3.2 [Assigning WAI link attribute values with dedicated formats](#) on page 652

§34.3.3 [Assigning WAI link attribute values with PI markers](#) on page 652



### 34.3.1 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 PI marker; and either may reference macro variable `<$$_linksrc>`.

### 34.3.2 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 DITA document, just before the link you would insert an element to which you have assigned format *Linkname* containing the name you want displayed for the link destination. 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.

### 34.3.3 Assigning WAI link attribute values with PI markers

You can use PI markers to provide text alternatives (and other attributes) for links. The attribute applies to the next link after the PI marker. The PI 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, place a **LinkTitle** marker just before the link.

To assign a CSS class to a link, see §28.2.2.1 [Assigning a link class with a PI marker](#) on page 546.

## 34.4 Applying WAI markup to tables

*In this section:*

- §34.4.1 [Following WAI guidelines for tables](#) on page 653
- §34.4.2 [Choosing a WAI markup method for tables](#) on page 653
- §34.4.3 [Providing table summary and title information](#) on page 654
- §34.4.4 [Identifying table row and column information](#) on page 655

See also:

§35 [Identifying HTML table structure for WAI](#) on page 657

§36 [Marking HTML table cells for WAI](#) on page 667

### 34.4.1 Following WAI guidelines for tables

You can assign WAI attributes to tables, and within tables to rows, columns, and header cells. **DITA2Go** 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).

**DITA2Go** provides settings for the following attributes:

<code>abbr</code>	Abbreviation for the contents of a cell.
<code>axis</code>	Conceptual category of cell content, provided for queries.
<code>headers</code>	References to header-cell IDs.
<code>id</code>	ID (name) of a header cell.
<code>scope</code>	Rows or columns covered by a header cell.
<code>summary</code>	Description of a table's purpose and structure.
<code>title</code>	Brief description of table.

**DITA2Go** automatically generates markup for the following (non-WAI) attributes, based on row and column straddling in your tables:

<code>colspan</code>	Number of columns spanned (straddled) by a cell.
<code>rowspan</code>	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>

### 34.4.2 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 §34.1 [Comparing DITA2Go markup methods for WAI](#) on page 649). 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 DITA 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.

### 34.4.3 Providing table summary and title information

*In this section:*

§34.4.3.1 [Using a table attribute for summary or title](#) on page 654

§34.4.3.2 [Using a dedicated format for table summary or title](#) on page 654

§34.4.3.3 [Using a PI marker for table summary or title](#) on page 655

#### 34.4.3.1 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 is the TableID (see §4.3 [Identifying files and elements](#) on page 76).

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 §37 [Working with macros](#) on page 679 for more information.

#### 34.4.3.2 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, 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 *Tb/Tt* paragraph would become the text of the `title` attribute of the HTML `<table>` tag.

### 34.4.3.3 Using a PI marker for table summary or title

You can use a PI marker to provide a summary for a table, and another PI marker to provide a title. Each PI 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, place a **TableSummary** PI marker somewhere in the table, or just before the table.

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** PI marker includes characters `<` or `>`, these characters must be escaped, thus: `\<` or `\>`; otherwise the table might not be rendered correctly in HTML.

## 34.4.4 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 **DITA2Go** to generate WAI table-cell attributes are described in the following sections:

§35 [Identifying HTML table structure for WAI](#) on page 657

§36 [Marking HTML table cells for WAI](#) on page 667



# 35 Identifying HTML table structure for WAI

---

This section describes how to use **DITA2Go** configuration settings to identify table structure for WAI support. Topics include:

§35.1 [Identifying table rows and columns](#) on page 657

§35.2 [Associating table cells with header cells](#) on page 660

*See also:*

§34 [Generating WAI markup for HTML](#) on page 649

§36 [Marking HTML table cells for WAI](#) on page 667

## 35.1 Identifying table rows and columns

*In this section:*

§35.1.1 [Developing a strategy for row and column markup](#) on page 657

§35.1.2 [Comparing scope and id/headers accessibility methods](#) on page 657

§35.1.3 [Specifying a default accessibility method](#) on page 658

§35.1.4 [Overriding the default accessibility method](#) on page 659

### 35.1.1 Developing a strategy for row and column markup

**DITA2Go** 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 §35.1.3 [Specifying a default accessibility method](#) on page 658.
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 §35.2.1 [Specifying group properties for header cells](#) on page 660.
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 §35.2 [Associating table cells with header cells](#) on page 660.
4. Add `abbr` and `axis` attributes as needed; see §35.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 662 and §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666.

### 35.1.2 Comparing scope and id/headers accessibility methods

*scope method*    The `scope` method works well for simple tables. **DITA2Go** 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. <b>DITA2Go</b> 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.

### 35.1.3 Specifying a default accessibility method

*In this section:*

§35.1.3.1 [Establishing a basic policy for table accessibility](#) on page 658

§35.1.3.2 [Applying the scope method to all tables](#) on page 658

§35.1.3.3 [Applying the id/headers method to all tables](#) on page 659

#### 35.1.3.1 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, **DITA2Go** 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 §35.1.4 [Overriding the default accessibility method](#) on page 659.

If you specify group properties for some or all header cells (see §35.2.1 [Specifying group properties for header cells](#) on page 660), `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 §36 [Marking HTML table cells for WAI](#) on page 667. 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 PI marker, to apply the *same* method to an individual table; the result is duplicate attribute assignments. See §22.14.4 [Avoiding redundant attribute assignments in tables](#) on page 447.

#### 35.1.3.2 Applying the scope method to all tables

When `AccessMethod=Scope`, **DITA2Go** supplies the following WAI attributes for every table:

- `scope="colgroup"` for any header cells marked `ColGroup`; automatically sets `ColGroupElements=Yes` (see §33.3.2.3 [Enumerating table column groups](#) on page 628) if any column-header cells are so specified.
- `scope="rowgroup"` for any left-side cells marked `RowGroup`; automatically sets `HeadFootBodyTags=Yes` (see §33.3.2.4 [Wrapping table row groups](#) on page 629) 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 §36.2 [Using the scope method to identify table cells](#) on page 667 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 §33.3.2.4 [Wrapping table row groups](#) on page 629.

### 35.1.3.3 Applying the id/headers method to all tables

When `AccessMethod=IDheaders`, **DITA2Go** 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 §36.3 [Using the id/headers method to identify table cells](#) on page 669 for more information.

### 35.1.4 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 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:

§36.2 [Using the scope method to identify table cells](#) on page 667

§36.3 [Using the id/headers method to identify table cells](#) on page 669

To override attributes at the cell level, use one of the table-cell markup methods instead; see:

§35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661

§35.2.3 [Assigning table-cell attribute values with dedicated formats](#) on page 665

§35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666

**Note:** If you specify a default method for all tables (see §35.1.3 [Specifying a default accessibility method](#) on page 658), do not also use an override to apply the *same* method to an individual table; the result is duplicate attribute assignments. See §22.14.4 [Avoiding redundant attribute assignments in tables](#) on page 447.

## 35.2 Associating table cells with header cells

If specifying an accessibility method does not prove adequate for some or all tables in your document, **DITA2Go** 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 §36 [Marking HTML table cells for WAI](#) on page 667 for information about these settings.
- Attributes you specify for rows, columns, or individual cells via paragraph formats or PI markers; these apply to the tables in which you use them.

*In this section:*

§35.2.1 [Specifying group properties for header cells](#) on page 660

§35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661

§35.2.3 [Assigning table-cell attribute values with dedicated formats](#) on page 665

§35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666

### 35.2.1 Specifying group properties for header cells

*In this section:*

§35.2.1.1 [Defining blocks of header cells](#) on page 660

§35.2.1.2 [Using header cells to define column groups](#) on page 660

§35.2.1.3 [Using header cells to define row groups](#) on page 661

#### 35.2.1.1 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; **DITA2Go** 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 §36.5 [Using ColGroup and RowGroup cells](#) on page 676 for more information.

#### 35.2.1.2 Using header cells to define column groups

**DITA2Go** 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 §35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661)

- a **CellGroup** PI marker that you have given content `col` (see §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666).

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 §36.5.1 [Understanding how the ColGroup property works](#) on page 676 for more information.

### 35.2.1.3 Using header cells to define row groups

**DITA2Go** 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 §35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661)
- a **CellGroup** PI marker that you have given content `row` (see §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666).

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 §36.5.2 [Understanding how the RowGroup property works](#) on page 677 for more information.

## 35.2.2 Using paragraph formats for table-cell attributes

*In this section:*

- §35.2.2.1 [Choosing how to use paragraph formats for WAI markup](#) on page 661
- §35.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 662
- §35.2.2.3 [Assigning values to WAI attributes](#) on page 663
- §35.2.2.4 [Specifying a different HTML table-cell tag](#) on page 663
- §35.2.2.5 [Identifying table cells with formats: an example](#) on page 664

### 35.2.2.1 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 §34.1.2 [Using paragraph formats for WAI attributes](#) on page 649 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.

### 35.2.2.2 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 35-1](#) describes each of these properties.

**Table 35-1 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 (Scope, Abbr, and Axis) 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 §35.2.1.2 <a href="#">Using header cells to define column groups</a> on page 660.
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.
RowGroup	The cell is a header cell that starts a row group. See §35.2.1.3 <a href="#">Using header cells to define row groups</a> on page 661.
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 <code>[StyleCellScope]</code> .
Span	The cell is a header cell that applies to more than one column or row; <code>id="spanN"</code> is assigned to the cell, regardless of <code>ColSpanIDs</code> or <code>RowSpanIDs</code> settings.

### 35.2.2.3 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"
```

### 35.2.2.4 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 §33.3.3 [Identifying table headers and footers](#) on page 630.

### 35.2.2.5 Identifying table cells with formats: an example

Suppose your DITA document contains a table with the following structure; [Table 35-2](#) 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 35-2 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 an element that is assigned a paragraph format different from (or perhaps in addition to) the paragraph format assigned to elements used in ordinary row and column headers:

- Because it spans more than one column, the topmost column-header cell in [Table 35-2](#) 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 an element that is assigned a paragraph format different from the format for column headers.
- The rightmost column in [Table 35-2](#) 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 35-2](#):

```

[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 35-2](#) on page 664 looks something like this (omitting display attributes) in **DITA2Go**-generated HTML:

```
<table>
<caption><p>Table 35-2: 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>
```

### 35.2.3 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]`.

For example, suppose you assign paragraph format *WAIabbr* to an element, 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 an element assigned format *WAIabbr* and give it content *Type*. 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 §30.2.6 [Eliminating unwanted paragraphs](#) on page 569.

### 35.2.4 Assigning table-cell attribute values with PI markers

You can use PI markers to apply WAI cell attributes. Each PI marker name must start with **Cell** and end with the name of an attribute. The PI markers for table cells are as follows:

<b>CellAbbr</b>	Abbreviation for content of a cell; adds <code>abbr</code> attribute.
<b>CellAxis</b>	Conceptual category for the content of a cell; adds <code>axis</code> attribute.
<b>CellID</b>	Cell identifier; replaces the value of any generated <code>id</code> attribute.
<b>CellScope</b>	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, place a **CellAbbr** PI marker in the cell.

Two additional PI 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:

<b>CellGroup</b>	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 §35.2.1 <a href="#">Specifying group properties for header cells</a> on page 660.
<b>CellSpan</b>	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 §35.2.2.2 [Assigning WAI attributes to paragraph formats](#) on page 662.

**Note:** If you specify a default access method for all tables (see §35.1.3 [Specifying a default accessibility method](#) on page 658), do not also use a PI marker to apply the *same* method to individual tables; the result is duplicate attribute assignments. See §22.14.4 [Avoiding redundant attribute assignments in tables](#) on page 447.

# 36 Marking HTML table cells for WAI

This section describes how to use **DITA2Go** configuration settings to fine-tune the association of table-cell content with row- and column-header information. Topics include:

§36.1 [Understanding table cell settings](#) on page 667

§36.2 [Using the scope method to identify table cells](#) on page 667

§36.3 [Using the id/headers method to identify table cells](#) on page 669

§36.4 [Overriding default table-cell settings](#) on page 675

§36.5 [Using ColGroup and RowGroup cells](#) on page 676

See also:

§34 [Generating WAI markup for HTML](#) on page 649

§35 [Identifying HTML table structure for WAI](#) on page 657

## 36.1 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 referenced by a single ditamap, you can include those settings in a configuration file named after the ditamap file; for example, Chap2.ini. See §42.1 [Using a different configuration for selected files](#) on page 765.

**DITA2Go** 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, **DITA2Go** 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.

## 36.2 Using the scope method to identify table cells

[Table 36-1](#) lists the scope settings you can specify in the [Tables] section of the configuration file.

**Table 36-1** WAI scope attributes for table cells

	[Tables] setting	Default value	[TableAccess] override	Purpose
<b>Column</b>	ScopeCol	No	ScopeCol	Apply scope="col" (the default) to column-header cells
	ScopeColGroup	No	ScopeColGroup	Apply scope="colgroup" to ColGroup header cells*
<b>Row</b>	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 PI marker

Use these settings to identify column and row header cells that apply to more than one body column or row, either explicitly 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 §36.4 [Overriding default table-cell settings](#) on page 675.

**Note:** If you set AccessMethod=Scope, **DITA2Go** 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 §33.3.2.3 [Enumerating table column groups](#) on page 628.
- Specifying row groups automatically sets [Tables]HeadFootBodyTags=Yes; for more information, see §33.3.2.4 [Wrapping table row groups](#) on page 629.

The group scope attributes work in concert with ColGroup and RowGroup cells: header cells that are assigned [HTMLParaStyles] property ColGroup or RowGroup, described in §35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661; or that contain PI marker type **CellGroup**, described in §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666.

**Note:** If any of your tables have footer rows, when you use scope="rowgroup" the resulting HTML might contain some surprises; see §33.3.2.5 [Positioning table footer rows \(deprecated\)](#) on page 629 in §33.3.2.4 [Wrapping table row groups](#) on page 629.

## 36.3 Using the id/headers method to identify table cells

*In this section:*

- §36.3.1 [Choosing an id/headers level](#) on page 669
- §36.3.2 [Specifying id/headers attributes for table cells](#) on page 669
- §36.3.3 [Grouping header cells for identification](#) on page 670
- §36.3.4 [Column-group and row-group extent](#) on page 671
- §36.3.5 [Choosing a different row-group method](#) on page 672
- §36.3.6 [Using span attributes to identify rows and columns](#) on page 672
- §36.3.7 [Column-span and row-span extent](#) on page 673
- §36.3.8 [Identifying individual table cells by row and column](#) on page 674
- §36.3.9 [Column and row extent](#) on page 674
- §36.3.10 [Using span IDs with row or column IDs](#) on page 675

### 36.3.1 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 §35.2.3 [Assigning table-cell attribute values with dedicated formats](#) on page 665 or §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666.

### 36.3.2 Specifying id/headers attributes for table cells

[Table 36-2](#) 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 §36.4 [Overriding default table-cell settings](#) on page 675.

**Table 36-2 WAI id/header table cell attributes**

	[Tables] setting	Default value	Purpose	Ref.
<b>Column</b>	ColGroupHead	group	Name the id to use for column group headers	<a href="#">36.3.3</a>
	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	<a href="#">36.3.6</a>
	ColSpanIDs	No	Add id="spanN" to column-header cells designated Span via [HTMLParaStyles] or <b>CellSpan</b> PI marker, headers="spanN" to cells in the column	
	ColHead	col	Name the id to use for columns	<a href="#">36.3.8</a>
	ColIDs	No	Add id="colN" to column headers, headers="colN" to cells in the column	
<b>Row</b>	RowGroupHead	group	Name the id to use for row group headers	<a href="#">36.3.3</a>
	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	<a href="#">36.3.6</a>
	RowSpanIDs	No	Add id="spanN" to row-header cells (first cell in each row) designated Span via [HTMLParaStyles] (or via a <b>CellSpan</b> PI marker), headers="spanN" to cells in the row	
	RowHead	row	Name the id to use for rows	<a href="#">36.3.8</a>
	RowIDs	No	Add id="rowN" to leftmost column, headers="rowN" to cells in the row	
* Cells marked ColGroup or RowGroup via [HTMLParaStyles] <i>parafmt=xGroup</i> or via <b>CellGroup</b> PI marker				

### 36.3.3 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 §35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661; or that contain PI marker type **CellGroup**, described in §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666.

```
[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 **DITA2Go** 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, **DITA2Go** 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, **DITA2Go** generates the following identifiers:

id="groupN"

- for each ColGroup cell (header cell that contains either a paragraph designated [HTMLParaStyles] ColGroup, or a **CellGroup** PI 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 §36.3.8 [Identifying individual table cells by row and column](#) on page 674 for information about specifying IDs for individual columns. See also §36.5.1 [Understanding how the ColGroup property works](#) on page 676.

*Row groups*

When you specify RowGroupIDs=Yes, **DITA2Go** generates the following identifiers:

id="groupN"

- for each RowGroup cell (header cell that contains either a paragraph designated [HTMLParaStyles] RowGroup, or a **CellGroup** PI 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 §36.3.8 [Identifying individual table cells by row and column](#) on page 674 for information about specifying IDs for individual rows. See also §36.5.2 [Understanding how the RowGroup property works](#) on page 677.

### 36.3.4 Column-group and row-group extent

Figure 36-1 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 36-1 Extent of row and column groups**

	id=colgrp1		id=colgrp2	
id=rowgrp1				
id=rowgrp2				
id=rowgrp3				

headers=colgrp1

headers=rowgrp2

headers=colgrp1 rowgrp2

### 36.3.5 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.

### 36.3.6 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 **DITA2Go** 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 §35.2.2 [Using paragraph formats for table-cell attributes](#) on page 661; or that contain PI marker type **CellSpan**, described in §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666.

```
[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 CellSpan 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

```

**DITA2Go** 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** PI 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 §36.4 [Overriding default table-cell settings](#) on page 675.

#### Column-span and row-span identifiers

The values you specify for ColSpanHead and RowSpanHead are the names **DITA2Go** 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, **DITA2Go** 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, **DITA2Go** generates the following identifiers:

id="spanN"	for each column-header Span cell (cell containing a paragraph designated [HTMLParaStyles] Span, or a <b>CellSpan</b> PI marker).
headers="spanN"	for each cell in each column below (spanned by) the id="spanN" cell.

#### Row spans

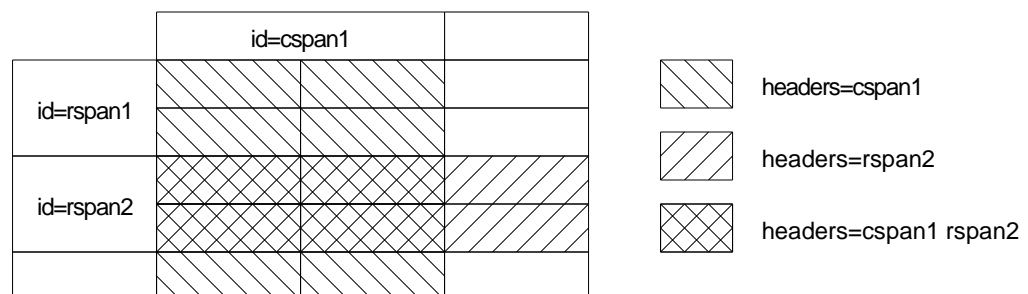
When you specify RowSpanIDs=Yes, **DITA2Go** generates the following identifiers:

id="spanN"	for each row-header Span cell (cell containing a paragraph designated [HTMLParaStyles] Span, or a <b>CellSpan</b> PI 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.

### 36.3.7 Column-span and row-span extent

[Figure 36-2](#) 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 36-2** Extent of column and row spans



### 36.3.8 Identifying individual table cells by row and column

When you use the following settings, **DITA2Go** 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 §36.4 [Overriding default table-cell settings](#) on page 675.

#### Column and row identifiers

The values you specify for `ColHead` and `RowHead` are the names **DITA2Go** 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`, **DITA2Go** uses the defaults: `col` and `row`, respectively.

#### Columns

When you specify `ColIDs=Yes`, **DITA2Go** 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
```

**DITA2Go** 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, **DITA2Go** generates `headers="col3"` for the body cells in columns 3, 4, and 5.

#### Rows

When you specify `RowIDs=Yes`, **DITA2Go** generates the following identifiers for each row:

```
id="rowN"           for the first (top left) cell in row n that does not already
                    contain an id attribute (such as id="col1" in the first
                    header row)
headers="rowN"      for each body cell in row n to the right of the id=rowN cell
```

### 36.3.9 Column and row extent

[Figure 36-3](#) 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 36-3 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

### 36.3.10 Using span IDs with row or column IDs

When ColIDs=Yes:

- If ColSpanIDs=No, **DITA2Go** 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, **DITA2Go** 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, **DITA2Go** 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, **DITA2Go** 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 §36.3.6 [Using span attributes to identify rows and columns](#) on page 672 for more information about RowSpanIDs.

## 36.4 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 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.
aa123456=ColIDs NoRowIDs
group5=ScopeColGroup
Format A=RowSpanIDs NoColIDs

```

## 36.5 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:*

[§36.5.1 Understanding how the ColGroup property works](#) on page 676

[§36.5.2 Understanding how the RowGroup property works](#) on page 677

See [§35.2.1 Specifying group properties for header cells](#) on page 660 for information about specifying ColGroup and RowGroup cells.

### 36.5.1 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 36-3](#) summarizes the effects of the ColGroup property when combined with these settings.

**Table 36-3 ColGroup property effects**

Setting	ColGroupElements	Yes				No			
	ColGroupIDs	Yes		No		Yes		No	
	scope="colgroup" *	Yes	No	Yes	No	Yes	No	Yes	No
Effect	starts new <colgroup>	Yes	Yes	Yes	Yes	No	No	No	No
	id/headers="groupN"	Yes	Yes	No	No	Yes	Yes	No	No
	scope attribute applied	Yes	No	Yes	No	No	No	No	No

\* Set via **CellScope** PI 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** PI 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 §36.2 [Using the scope method to identify table cells](#) on page 667.

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 36-1](#) on page 672), each get a matching headers="groupN" attribute. If ColGroupElements=Yes, these are the cells subsumed by the <colgroup> element. See §36.3.3 [Grouping header cells for identification](#) on page 670.

## 36.5.2 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 36-4](#) summarizes the effects of the RowGroup property when combined with these settings.

**Table 36-4 RowGroup property effects**

Setting	HeadFootBodyTags	Yes				No			
	RowGroupIDs	Yes		No		Yes		No	
	scope="rowgroup" *	Yes	No	Yes	No	Yes	No	Yes	No

\* Set via **CellScope** PI marker or [HTMLParaStyles]fmt=Scope, [StyleCellScope]fmt=rowgroup

**Table 36-4 RowGroup property effects**

<b>Effect</b>	starts new <tbody>	Yes	Yes	Yes	Yes	No	No	No	No
	id/headers="groupN"	Yes	Yes	No	No	Yes	Yes	No	No
	scope attribute applied	Yes	No	Yes	No	No	No	No	No

\* Set via **CellScope** PI 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** PI 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 §36.2 [Using the scope method to identify table cells](#) on page 667.

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 36-1](#) on page 672), until the next cell down that contains a RowGroup paragraph. See §36.3.3 [Grouping header cells for identification](#) on page 670.

# 37 Working with macros

---

You can use macros to insert any content into the output stream. Because the **DITA2Go** macro language is Turing-complete, the **DITA2Go** macro facility is powerful enough to let you insert anything in RTF output, and do almost anything to HTML or XML output. Topics include:

- §37.1 [Defining and invoking macros](#) on page 679
- §37.2 [Accessing DITA2Go macro libraries](#) on page 684
- §37.3 [Using macro variables](#) on page 687
- §37.4 [Using multiple-value list variables](#) on page 695
- §37.5 [Accessing settings with configuration macros](#) on page 699
- §37.6 [Using expressions in macros](#) on page 700
- §37.7 [Passing a parameter to a macro](#) on page 709
- §37.8 [Debugging macros](#) on page 709
- §37.9 [Deploying macros and macro variables](#) on page 710
- §37.10 [Using macros to fine-tune HTML or XML output](#) on page 713

*See also:*

- §42.2.3 [Overriding settings with macros](#) on page 767

## 37.1 Defining and invoking macros

*In this section:*

- §37.1.1 [Defining macros](#) on page 679
- §37.1.2 [Invoking a macro](#) on page 683
- §37.1.3 [Nesting macros](#) on page 683
- §37.1.4 [Using predefined macros](#) on page 683

### 37.1.1 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 §37.1.1.2 [Understanding where you can define named macros](#) on page 680.

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:*

- §37.1.1.1 [Understanding what a macro definition can include](#) on page 680
- §37.1.1.2 [Understanding where you can define named macros](#) on page 680
- §37.1.1.3 [Escaping special characters in macro definitions](#) on page 680
- §37.1.1.4 [Managing line breaks in macro definitions](#) on page 681
- §37.1.1.5 [Including comments in macro definitions](#) on page 681
- §37.1.1.6 [Obtaining RTF code for macro definitions](#) on page 682



### 37.1.1.1 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.
- 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 §43.1.5 [Supplying system commands in a macro](#) on page 779.

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 **DITA2Go** 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.

### 37.1.1.2 Understanding where you can define named macros

You can put **DITA2Go** macro definitions in any of the following places:

- **Best place:** in a macro library file; see §37.2 [Accessing DITA2Go macro libraries](#) on page 684.
- **If large or complex:** individually in separate macro files; see §37.2.3 [Storing a macro definition in a separate file](#) on page 685.
- **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 §37.1.2 [Invoking a macro](#) on page 683).

*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 §39.5.2 [Deciding what to include in a general configuration template](#) on page 742).

*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 **DITA2Go** 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.

### 37.1.1.3 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 §37.1.1.5 [Including comments in macro definitions](#) on page 681.

#### 37.1.1.4 Managing line breaks in macro definitions

A macro definition does not have to be all on one line; **DITA2Go** 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.

#### 37.1.1.5 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 <$$currpath>\help ugdita2go
; But the following line will be executed:
jhindex <$$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 §37.1.1.3 [Escaping special characters in macro definitions](#) on page 680). 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.

### 37.1.1.6 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 DITA2Go](#)

[Generate RTF code with DITA2Go.](#)

*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 **DITA2Go** 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  
DITA2Go*

Another way to obtain RTF code is to create an example in DITA, run **DITA2Go**, and then copy/paste the resulting RTF code into your `d2rtf.ini` configuration file or into a macro library file. **DITA2Go** produces RTF output that is even more readable than the output from `pprtf.exe`.

*Generate RTF  
code with  
DITA2Go*

For paragraphs, you can use CodeStore to generate RTF code; see §37.3.5.2 [Inserting code with the CodeStore property](#) on page 693.

## 37.1.2 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 **DITA2Go** 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 §37.9.1 [Understanding where to use macros and macro variables](#) on page 710.
- in an **HTML Macro** PI marker; see §37.9.5 [Using HTML Macro PI markers to invoke macros](#) on page 713.

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*

**DITA2Go** ignores the invocation of any macro for which no definition can be found, unless you specify a special debugging option; see §37.8 [Debugging macros](#) on page 709. 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 §37.2.4 [Including macro definitions in your own macro library](#) on page 685.

## 37.1.3 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, **DITA2Go** 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 **DITA2Go** would continue. You cannot crash it by making it loop.

## 37.1.4 Using predefined macros

**DITA2Go** provides several predefined macros for HTML, listed in [Table 37-1](#), and for RTF, listed in [Table 37-2](#).

*Predefined macro  
names are  
reserved*

Avoid giving any of your own macros a name that starts with an underscore; the **DITA2Go** definition takes precedence. The “\$” says “this is a **DITA2Go** construct”; the “\_” says “the name is reserved, not one of yours”.

**Table 37-1 Predefined macros for HTML output**

Macro	Description	Ref.
<code>&lt;\$_TopicStartCode&gt;</code>	Macros from marker type <b>TopicStartCode</b>	<a href="#">38.2</a>
<code>&lt;\$_trail&gt;</code>	Inserts a “breadcrumb trail” of links	<a href="#">29.2</a>

The predefined macros for RTF output, listed in [Table 37-2](#), are intended for use in defining RTF output formats; see §7.4.8 [Assigning content-adding properties to formats](#) on page 118.

**Table 37-2 Predefined macros for RTF output**

Macro	Description
<\$_style( <i>stylename</i> )>	RTF start code for a paragraph or character format
<\$_colornum( <i>colorref</i> )>	Color number, after \\cf or \\cb
<\$_fontnum( <i>fontname</i> )>	Font number, after \\f
<\$_stylenum( <i>stylename</i> )>	Style number; after \\s i
<\$_styleref( <i>stylename</i> )>	STYLEREF field with the named style
<\$_pageref>	PAGEREF field referencing last bookmark used

## 37.2 Accessing DITA2Go macro libraries

*In this section:*

- §37.2.1 [Understanding DITA2Go-supplied macro libraries](#) on page 684
- §37.2.2 [Modifying DITA2Go-supplied macro definitions](#) on page 685
- §37.2.3 [Storing a macro definition in a separate file](#) on page 685
- §37.2.4 [Including macro definitions in your own macro library](#) on page 685

### 37.2.1 Understanding DITA2Go-supplied macro libraries

Your **DITA2Go** distribution includes several macro library files in the form of macro configuration templates, listed in [Table 39-9](#). These macro libraries are located in directory %OMSYSHOME%\d2g\macros. The templates are chained together by references.

**Table 37-3 Dita2Go macro libraries**

Macro library	Scope of macro settings	Referenced template
d2g_macros.ini	All <b>DITA2Go</b> projects	(None)
d2htm_macros.ini	<b>DITA2Go</b> HTML/XML projects	d2g_macros.ini
d2rtf_macros.ini	<b>DITA2Go</b> RTF projects	d2g_macros.ini

To access a macro library (for example):

```
[Templates]
; Macros = path to macro library file
Macros = %OMSYSHOME%\d2g\d2htm_macro.ini
```

**DITA2Go** 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 **DITA2Go** finds a repeat.

Your **DITA2Go** project configuration file should reference d2htm\_macros.ini or d2rtf\_macros.ini, either directly or indirectly through your own macro library file.

*See also:*

- §39.2 [Referencing configuration files and templates](#) on page 731
- §37.2.4.3 [Creating a chain of macro libraries](#) on page 687

## 37.2.2 Modifying DITA2Go-supplied macro definitions

You can modify the macro definitions included in the macro libraries supplied with your **DITA2Go** distribution, located in %OMSYSHOME%\d2g\macros. However, if you change anything in those files, whenever you update **DITA2Go** you will need to run a file comparison program to see if anything has been added or changed by Omni Systems developers; see §1.3.9 [Obtain a file comparison tool \(optional\)](#) on page 35.

An alternative is to create your own macro library file and copy into it any macros you want to alter; see §37.2.4 [Including macro definitions in your own macro library](#) on page 685.

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 §37.3 [Using macro variables](#) on page 687):

```
[Spacer]
" alt="[spacer]">

[MacroVariables]
spacerwidth=80
```

You can copy these definitions into your own macro library file, and modify them as you wish.

## 37.2.3 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 **DITA2Go** 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:

§37.2.4 [Including macro definitions in your own macro library](#) on page 685

## 37.2.4 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, **DITA2Go** looks for the definition in a macro library file.

*In this section:*

- §37.2.4.1 [Creating a macro library](#) on page 686
- §37.2.4.2 [Creating a file-specific macro library](#) on page 686
- §37.2.4.3 [Creating a chain of macro libraries](#) on page 687

### 37.2.4.1 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 **DITA2Go** projects.
2. Add macro definitions to the file, each in its own section, as described in §37.1.1 [Defining macros](#) on page 679.
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 `d2rtf_macros.ini` or `d2htm_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%\d2g\d2htm_macro.ini
```

In your project configuration file:

```
[Templates]
Macros = D:\MacroLibs\MyMacros.ini
```

If you omit a path, **DITA2Go** 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, **DITA2Go** looks in `%OMSYSHOME%\d2g\macros` for a file named `d2htm_macros.ini` or `d2rtf_macros.ini`.

### 37.2.4.2 Creating a file-specific macro library

If you store macro definitions in a file named the same as the DITA file you are converting, but with extension `.ini`, **DITA2Go** 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.



### 37.2.4.3 Creating a chain of macro libraries

A macro library file can include a setting for `[Templates]Macros`, so the chain of libraries for **DITA2Go** to search for macro definitions can be any length. However, all files in the chain must have distinct names; the chain stops if **DITA2Go** 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 map configuration file (see §42.1 [Using a different configuration for selected files](#) on page 765) overrides a definition in the project configuration file, *for that map only*.

**DITA2Go** builds a set of macros for each DITAmapping file in your project by starting with the most specific macro definitions: those in the `mapfile.ini` configuration file, if there is one. Next come macro definitions in your project configuration file.

Next, if `mapfile.ini` includes a value for `[Templates]Macros`, definitions in the referenced macro library (and any additional libraries chained to it) are applied. If `mapfile.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 `mapfile.ditamap` either from `mapfile.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.

## 37.3 Using macro variables

*In this section:*

§37.3.1 [Creating and invoking macro variables](#) on page 687

§37.3.2 [Assigning values to macro variables](#) on page 688

§37.3.3 [Incrementing and decrementing macro variables](#) on page 690

§37.3.4 [Using predefined macro variables](#) on page 691

§37.3.5 [Creating macro variables from paragraph content](#) on page 692

### 37.3.1 Creating and invoking macro variables

*In this section:*

§37.3.1.1 [Naming macro variables](#) on page 687

§37.3.1.2 [Creating a macro variable](#) on page 688

§37.3.1.3 [Invoking a macro variable](#) on page 688

#### 37.3.1.1 Naming macro variables

A **DITA2Go** macro variable name looks like a **DITA2Go** 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 **DITA2Go**, and cannot be used for other purposes; see §37.3.4 [Using predefined macro variables](#) on page 691. The name of a *predefined* **DITA2Go** 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 **DITA2Go** definition takes precedence. The “\$\$” says “this is a **DITA2Go** macro variable”; the “\_” says “the name is reserved, not one of yours”.

A **DITA2Go** macro variable is not the same as a **DITA2Go** variable, though they share the same naming convention. A **DITA2Go** macro variable overrides a **DITA2Go** variable of the same name. See §12.1 [Understanding how DITA2Go user variables work](#) on page 185.

### 37.3.1.2 Creating a macro variable

You create a **DITA2Go** 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 §37.3.2 [Assigning values to macro variables](#) on page 688.
- List the name of the variable in [MacroVariables] (for use in macros); see §37.3.2 [Assigning values to macro variables](#) on page 688.
- Assign a TextStore or CodeStore property to a paragraph format; see §37.3.5 [Creating macro variables from paragraph content](#) on page 692.

### 37.3.1.3 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 §37.6.3 [Displaying expression results in output](#) on page 702.

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>
```

## 37.3.2 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 **DITA2Go** macros:

```
<$$varname = $$othername>
<$$varname = (expr)> (See §37.6 Using expressions in macros on page 700)
<$$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 37-4](#).

**Table 37-4 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 §37.6.3 [Displaying expression results in output](#) on page 702.

*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 §37.6.7 [Using indirection in expressions](#) on page 708.

*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:

§37.4.2 [Assigning a value to a list-variable item](#) on page 696

§42.2.4 [Assigning values to configuration variables](#) on page 768

### 37.3.3 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 DITA 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++>
```

*Increment by assignment*

You can use an assignment (see §37.3.2 [Assigning values to macro variables](#) on page 688) 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 §37.6.3 [Displaying expression results in output](#) on page 702 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 §37.6.7 [Using indirection in expressions](#) on page 708.

### 37.3.4 Using predefined macro variables

DITA2Go provides a collection of predefined macro variables, listed in [Table 37-5](#). 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 §37.6.1 [Understanding macro expressions](#) on page 700).
- Format output of a predefined macro variable; see §37.6.3 [Displaying expression results in output](#) on page 702).

**Note:** Only `<$$_basename>` and `<$$_currpath>` can be used in system commands; other predefined macro variables do not work in system commands. See §43.1.5 [Supplying system commands in a macro](#) on page 779.

**Table 37-5** Predefined macro variables

Macro variable	Where used	Description	Ref
<code>\$\$_basefile</code>	HTML split files	Base name only of parent file, without extension	<a href="#">27.7</a>
<code>\$\$_basename</code>	System commands	Base file name (without path or extension) of current DITA file or map	<a href="#">43.1.2</a>
<code>\$\$_basetitle</code>	HTML split files	Original document title, unaffected by splits	<a href="#">27.7</a>
<code>\$\$_class</code>	Elements (HTML)	CSS class name of current paragraph	<a href="#">37.6.6</a>

**Table 37-5 Predefined macro variables (continued)**

Macro variable	Where used	Description	Ref
\$\$_count	Loop constructs	Current iteration count for <\$_repeat> loops	<a href="#">37.6.4.3</a>
\$\$_currbase	Output files	File name of current file, without extension	<a href="#">27.7</a>
\$\$_currfile	Output files	File name of current file, with extension,	<a href="#">27.7</a>
\$\$_currfilepath	Output files	Path and name of current file, with extension	<a href="#">27.7</a>
\$\$_currpath	System commands	Path, without trailing slash, to project directory	<a href="#">43.1.2</a>
\$\$_currtitle	HTML split files	Current file title, unaffected by extracts	<a href="#">27.7</a>
\$\$_dcount	Loop constructs	Down-count for <\$_repeat> loops	<a href="#">37.6.4.3</a>
\$\$_element	Elements (HTML)	Name of the current element	<a href="#">37.6.6</a>
\$\$_extrfile	HTML extract files	File name of extracted file	<a href="#">27.8.3</a>
\$\$_extrgraph	HTML extract files	File name of first extracted graphic	<a href="#">27.8.3</a>
\$\$_extrtitle	HTML extract files	Title of extracted file	<a href="#">27.8.3</a>
\$\$_firstfile	HTML split files	1 if first split part after original file, otherwise 0	<a href="#">27.7</a>
\$\$_graphbase	HTML graphics	File name for <img src= /> attribute, no extension	<a href="#">32.4.2</a>
\$\$_graphorighigh	HTML graphics	Original height in pixels of the image	<a href="#">32.4.2</a>
\$\$_graphorigwide	HTML graphics	Original width in pixels of the image	<a href="#">32.4.2</a>
\$\$_graphsrc	HTML graphics	File name for <img src= /> attribute, with extension	<a href="#">32.4.2</a>
\$\$_indexfilename	HTML index	File name for non-Help HTML index file, with extension	<a href="#">14.8.6.4</a>
\$\$_lastfile	HTML split files	1 if last part (regardless of splitting), or if unsplit; otherwise 0	<a href="#">27.7</a>
\$\$_linksrc	HTML link attributes	href content of a link	<a href="#">28.2.4</a>
\$\$_macroparam	Macros	Value of parameter passed in parentheses	<a href="#">37.7</a>
\$\$_nextfile	HTML split files	File name of split part that follows \$\$_currfile	<a href="#">27.7</a>
\$\$_nexttitle	HTML split files	Title of \$\$_nextfile split part	<a href="#">27.7</a>
\$\$_paratag	Formats	Name of current paragraph format	<a href="#">37.6.6</a>
\$\$_prevfile	HTML split files	File name of split part that precedes \$\$_currfile	<a href="#">27.7</a>
\$\$_prevtitle	HTML split files	Title of \$\$_prevfile split part	<a href="#">27.7</a>
\$\$_prjpath	System commands	Path (without trailing slash) to the directory where the map file resides	<a href="#">43.1</a>
\$\$_tblcols	Tables	Count of columns in current table	<a href="#">33.6.6</a>
\$\$_tblrows	Tables	Count of rows in current table	<a href="#">33.6.6</a>
\$\$_wcount	Loop constructs	Iteration count for <\$_while> loops	<a href="#">37.6.4.3</a>

### 37.3.5 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:*

§37.3.5.1 [Capturing paragraph content with the TextStore property](#) on page 692

§37.3.5.2 [Inserting code with the CodeStore property](#) on page 693

§37.3.5.3 [Understanding why TextStore and CodeStore work differently](#) on page 694

#### 37.3.5.1 Capturing paragraph content with the TextStore property

To store the text content of a 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 TextStore property to a paragraph format, but you do not supply a macro variable name in section [StyleTextStore], <b>DITA2Go</b> uses the paragraph format name itself as the macro variable name.
<i>Plain text</i>	TextStore 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 Delete property to the paragraph format.
<i>Only last instance counts</i>	If more than one instance of a TextStore paragraph format appears in a portion of your document destined for a given split or extract file, the TextStore 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 TextStore 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 CodeStore paragraphs, which must <i>precede</i> the point of use (see §37.3.5.3 <a href="#">Understanding why TextStore and CodeStore work differently</a> on page 694).
<i>Content is persistent</i>	The content of a TextStore 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, <b>DITA2Go</b> 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.

### 37.3.5.2 Inserting code with the CodeStore property

To store the content of a 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.



<i>Format name = variable name</i>	If you assign the <code>CodeStore</code> property to a paragraph format, but you do not supply a macro variable name in <code>[StyleCodeStore]</code> , <b>DITA2Go</b> uses the paragraph format name itself as the macro variable name.
<i>Must precede point of use</i>	A paragraph with the content you want to appear at a certain point in the output must be the last instance in your DITA document that <i>precedes</i> 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 <b>DITA2Go</b> inserts the latest value in the output. You can give the macro variable a starting value by defining its name in section <code>[MacroVariables]</code> ; see §37.3.2 <a href="#">Assigning values to macro variables</a> on page 688.
<i>Content is ignored</i>	When you assign the <code>CodeStore</code> property to a paragraph format, <b>DITA2Go</b> removes all instances of text in that format from the output. You can restore the text by invoking the macro variable in a <code>CodeAfter</code> macro.  Observe the following caveats: <ul style="list-style-type: none"> <li>• Do not assign property <code>Delete</code> to the <code>CodeStore</code> paragraph format; if you do, the paragraph content will <i>not</i> be stored in the macro variable.</li> <li>• Avoid assigning <code>CodeStore</code> to a paragraph format that has any of the HTML <code>ListN</code> properties; the <code>&lt;/li&gt;</code> coding will be misplaced.</li> </ul>
<i>Insert HTML navigation links</i>	Suppose you have a paragraph format named <i>Nextsect</i> that you use for cross references to other DITA files. And suppose you want to save the text of any <i>Nextsect</i> paragraph in macro variable <code>&lt;\$\$Footnext&gt;</code> , so you can make the link appear in a footer in the HTML output. You would specify these settings: <pre> [HTMLParaStyles] Nextsect=CodeStore  [StyleCodeStore] Nextsect=Footnext </pre> <p>The content of each successive paragraph in format <i>Nextsect</i> would be stored in macro variable <code>&lt;\$\$Footnext&gt;</code>, replacing the previous value for each instance of <i>Nextsect</i>. Wherever you insert macro variable <code>&lt;\$\$Footnext&gt;</code>, its current content, taken from the latest instance of <i>Nextsect</i>, appears in the output.</p>
<i>Generate RTF code</i>	You can use <code>CodeStore</code> 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 your DITA document, using a special paragraph format (for example, <i>Copyr</i> ). Make the <i>Copyr</i> paragraphs conditional for “Help only” to keep the notices out of print versions of your document. Then specify the following configuration settings: <pre> [HelpStyles] Copyr=CodeStore  [Inserts] TopicEnd=&lt;\$\$Copyr&gt; </pre> <p>This is much easier than trying to compose RTF code yourself to insert via macro:</p> <pre> [Copyr] \pard \s12 \f3 \fs20 \b Copyright \'a9 2012 Softworks Inc. \par </pre> <p>Besides, code such as <code>\s12</code> and <code>\f3</code> could change from one run to the next.</p>

### 37.3.5.3 Understanding why `TextStore` and `CodeStore` work differently

**DITA2Go** DCL “write” filters, such as `dwhtml.dll`, operate in two phases:

- Scan phase:* The results of converting XML to DCL are stored in linked lists in memory.

	<i>Write phase:</i> The linked lists in memory are traversed, additional information is collected, and output files are written.
<i>TextStore</i>	TextStore information is set during scan phase, and is available during write phase.
<i>CodeStore</i>	CodeStore information is set during write phase, and is available only after the point in the DITA file where the CodeStore 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. TextStore processing is able to save the text content of a paragraph during scan phase, because the text content is known at that time. The TextStore property excludes items that are not known, such as links. This is the same mechanism used to generate Title content.
<i>Write phase</i>	During write phase, <b>DITA2Go</b> 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. CodeStore processing saves the coded text from a paragraph assigned the CodeStore property; however, at that point previous paragraphs have already been written to final output, and cannot be altered.

## 37.4 Using multiple-value list variables

In addition to single-value macro variables (see §37.3 [Using macro variables](#) on page 687), 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:*

- §37.4.1 [Understanding list-variable syntax](#) on page 695
- §37.4.2 [Assigning a value to a list-variable item](#) on page 696
- §37.4.3 [Initializing list variables](#) on page 696
- §37.4.4 [Using macros to process lists](#) on page 696
- §37.4.5 [Using pointers to process lists](#) on page 697
- §37.4.6 [Using a list instead of a conditional expression](#) on page 698

### 37.4.1 Understanding list-variable syntax

To create a list variable, all you have to do is use a **DITA2Go** 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[ ]>`.

### 37.4.2 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 §37.3.2 [Assigning values to macro variables](#) on page 688):

```
<$$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 15<sup>th</sup> 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).

### 37.4.3 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 §37.4.2 [Assigning a value to a list-variable item](#) on page 696).

Suppose you set `<$$MyList[1]=Second>` in a macro. If this macro or another macro subsequently refers to `<$$MyList[1]>` while **DITA2Go** is processing the same DITA file, the value is `Second`, not `First`. But the value in the configuration file does not change, so when **DITA2Go** processes the next DITA 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 DITA file, but is not stored for use in the next.

### 37.4.4 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 DITA file that **DITA2Go** splits into named pages (see §27 [Splitting and extracting files](#) on page 523). 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:

```
[DITA_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 ($$_currbase is $$DITA_File[$$val])>\
    <p class="SidebarTxt"><$$SideTitle[$$val]></p>\
  <$_else>\
    <p class="SidebarTxt"><a class="SidebarLnk"\
      href="<$$DITA_File[$$val]>.htm"><$$SideTitle[$$val]></a></p>\
    <$_endif>
  <$$val++>
<$_endwhile>
```

### 37.4.5 Using pointers to process lists

The method described in §37.4.4 [Using macros to process lists](#) on page 696 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[$$val]` and `*$txtptr[$$val]` indicate that these list variables are actually being used indirectly, as pointers to other lists. When you use the form `*$ptr[$$index]`, **DITA2Go** converts the reference internally to `$$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=$$DITA_File
2=$$SideTitles
```

To access an item in `[DITA_File]`, first assign to a macro variable the pointer-list item that points to `[DITA_File]`:

```
<$$ptr=$$PtrList[1]>
```

Thereafter you can use the macro variable as a pointer to `[DITA_File]`; and so, referring to the `[DITA_File]` list in §37.4.4 [Using macros to process lists](#) on page 696, `<*$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.

## 37.4.6 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 §37.6.4.2 [Using conditional expressions](#) on page 704) 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.

## 37.5 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.

§37.5.1 [Understanding configuration macros and variables](#) on page 699

§37.5.2 [Determining the value of a configuration variable](#) on page 699

§37.5.3 [Deploying configuration macros](#) on page 700

### 37.5.1 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:

<i>Section</i>	Name of a configuration-file section.
<i>Key</i>	Keyword, format name, or other identifier that appears to the left of the equals sign in a configuration setting under <i>[Section]</i> .

A *configuration macro* is a **DITA2Go** 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 §42.2.4 [Assigning values to configuration variables](#) on page 768.

### 37.5.2 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 PI marker or another macro, the override, not the original value, is the value returned for `<$$[section]key>`. See §42.2 [Overriding settings with PI markers or macros](#) on page 766.

*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 `<$$[section]key>` refers to an invalid configuration-file section, or to an invalid key, **DITA2Go** returns 0 (zero), which is interpreted as *false* in a conditional expression `<$_if($$[section]key ...)>`; see §37.6.4.2 [Using conditional expressions](#) on page 704.

### 37.5.3 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 add a new setting with a configuration variable, see §42.2.5 [Adding a new configuration setting on the fly](#) on page 768.

## 37.6 Using expressions in macros

In **DITA2Go** 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 **DITA2Go** macro expression is a strong value.

*In this section:*

§37.6.1 [Understanding macro expressions](#) on page 700

§37.6.2 [Understanding operands and operators](#) on page 701

§37.6.3 [Displaying expression results in output](#) on page 702

§37.6.4 [Using control structures in expressions](#) on page 704

§37.6.5 [Specifying substrings in expressions](#) on page 706

§37.6.6 [Using list variables in expressions](#) on page 707

§37.6.7 [Using indirection in expressions](#) on page 708

§37.6.8 [Removing spaces from strings: an example](#) on page 709

### 37.6.1 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.

**DITA2Go** 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}>
```



## 37.6.2 Understanding operands and operators

*Operands for  
macro  
expressions*

An operand can be any of the following:

- a macro variable: `$$name` (see §37.3 [Using macro variables](#) on page 687)
- 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 **DITA2Go** string operators; [Table 37-6](#) 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 §37.6.4.2 [Using conditional expressions](#) on page 704, and the unary string operators described in §37.6.5 [Specifying substrings in expressions](#) on page 706.

**Table 37-6 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>!=, &lt;&gt;</code>	not equal to	
	<code>&lt;</code>	less than	
	<code>&lt;=</code>	less than or equal to	
	<code>&gt;</code>	greater than	
	<code>&gt;=</code>	greater than or equal to	
Logical	<code>and, &amp;&amp;</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, &amp;&amp;!</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>&amp;</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 &gt;&gt; 8) &amp; 0xFF)</code> extracts the second-up byte from a number.
	<code>&amp;~</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>&lt;&lt; N</code>	shift first operand to the left <i>N</i> bits	
	<code>&gt;&gt; N</code>	shift first operand to the right <i>N</i> bits	

**Table 37-6 Operators for HTML macro expressions (continued)**

Type	Operator	Meaning	Comments
Arithmetic	+	plus	These are the usual suspects. The result of (n / 0) or (n % 0) is 0 (zero), because infinity is hard to represent.
	-	minus	
	*	times	
	/	divided by	
	%	modulo	
String	is	equal to	<b>is</b> and <b>is not</b> are caseless compares using <code>strcmp()</code> <b>plus</b> is like <code>strcat()</code> <b>before</b> and <b>after</b> use <code>strstr()</code> to find the 2nd operand in the 1st: (doggie <b>before</b> gi) <b>is</b> dog You can get a <code>strnicmp()</code> effect using "first <i>N</i> " or "last <i>N</i> " with "is" or "is not": (( <code>\$\$myvar</code> <b>first</b> 3) <b>is</b> ( <code>\$\$yourvar</code> <b>last</b> 3))
	is not	not equal to	
	plus	concatenated with	
	before	substring before the 1st (leftmost) occurrence of 2nd string in 1st	
	after	substring after the first (leftmost) occurrence of 2nd string in 1st	
	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	
	starts <code>\$\$str</code>	true if <code>\$\$str</code> is at the start	
	ends <code>\$\$str</code>	true if <code>\$\$str</code> is at the end	
	contains <code>\$\$str</code>	true if <code>\$\$str</code> occurs anywhere in the string	Integer result Boolean result Boolean result Boolean result First (leftmost) character is number 1 Default value of <i>N</i> is 1 Default value of <i>N</i> is 1 Default value of <i>N</i> is 1 converts <code>\$\$str</code> to lowercase converts <code>\$\$str</code> to uppercase converts each instance of <code>\$\$str1</code> in <code>\$\$str</code> to <code>\$\$str2</code>
	char <i>N</i>	<i>N</i> th character, counting from left	
	trim first <i>N</i>	all but first <i>N</i> characters	
	trim last <i>N</i>	all but last <i>N</i> characters	
	<code>\$\$str</code> lower	converts <code>\$\$str</code> to lowercase	
	<code>\$\$str</code> upper	converts <code>\$\$str</code> to uppercase	
	<code>\$\$str</code> replace <code>\$\$str1</code> with <code>\$\$str2</code>	converts each instance of <code>\$\$str1</code> in <code>\$\$str</code> to <code>\$\$str2</code>	
Conditional	?	"if" the 1st operand is true, "then" the 2nd operand is the value of the expression	<\$( <code>\$\$myvar</code> ? "yes" : "no")> is equivalent to:
	:	"else" the 3rd operand is the value of the expression	<\$_if ( <code>\$\$myvar</code> )> yes <\$_else> no <\$_endif>

### 37.6.3 Displaying expression results in output

In general, **DITA2Go** 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 §37.3.2 [Assigning values to macro variables](#) on page 688)
- *control statements*, which have no obvious meaning of their own (see §37.6.4 [Using control structures in expressions](#) on page 704).

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 37-7](#).

#### 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 37-7 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](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>.

## 37.6.4 Using control structures in expressions

**DITA2Go** 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:*

§37.6.4.1 [Understanding control-structure elements](#) on page 704

§37.6.4.2 [Using conditional expressions](#) on page 704

§37.6.4.3 [Using loop structures](#) on page 705

### 37.6.4.1 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 37-8](#); the **DITA2Go** control-structure definition takes precedence.

**Table 37-8** Predefined control-structure elements

Control element	Where used	Purpose	Ref.
<\$_break>	Loop structure	Skip to the end of a loop	<a href="#">37.6.4.3</a>
<\$_continue>	Loop structure	Jump back to the start of the next iteration	<a href="#">37.6.4.3</a>
<\$_else>	Conditional expression	Introduce a final alternate condition	<a href="#">37.6.4.2</a>
<\$_elseif>	Conditional expression	Introduce an intermediate alternate condition	<a href="#">37.6.4.2</a>
<\$_endif>	Conditional expression	End a conditional expression	<a href="#">37.6.4.2</a>
<\$_endrepeat>	Loop structure	End a count-down loop	<a href="#">37.6.4.3</a>
<\$_endwhile>	Loop structure	End a logical loop	<a href="#">37.6.4.3</a>
<\$_if>, <\$_if not>	Conditional expression	Begin a conditional expression	<a href="#">37.6.4.2</a>
<\$_repeat>	Loop structure	Begin a count-down loop	<a href="#">37.6.4.3</a>
<\$_until>	Loop structure	Begin a logical loop	<a href="#">37.6.4.3</a>
<\$_while>	Loop structure	Begin a logical loop	<a href="#">37.6.4.3</a>

### 37.6.4.2 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 §37.4.6 [Using a list instead of a conditional expression](#) on page 698.)

*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 §37.1.3 [Nesting macros](#) on page 683):

```
[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.

### 37.6.4.3 Using loop structures

DITA2Go 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 <code>&lt;\$_wcount&gt;</code> contains the loop count, starting with 1 (one).
<i>“Until” loops</i>	<p>Because “until” is really the same as “while not”, the <code>&lt;\$_while&gt;</code> runaway-prevention limit also applies to <code>&lt;\$_until&gt;</code> loops:</p> <pre>[HtmlOptions] WhileMax=128</pre> <p>For example, a loop controlled by <code>&lt;\$_until (0)&gt;</code> goes to the <code>WhileMax</code> limit, unless you include an effective <code>&lt;\$_break&gt;</code>. Predefined variable <code>&lt;\$_wcount&gt;</code> contains the loop count, starting with 1 (one).</p>
<i>“Repeat” loops</i>	<p>For <code>&lt;\$_repeat&gt;</code>, the runaway-prevention limit comes into play only if the count is set to zero:</p> <pre>[Macros] ; RepeatMax = maximum count for &lt;\$_repeat&gt; when value is not given, so ; that loop continues until a &lt;\$_break condition&gt; is met RepeatMax=128</pre> <p>The current loop count is held in predefined variable <code>&lt;\$_count&gt;</code>, and the down-count starting with <code>expr</code> is in predefined variable <code>&lt;\$_dcount&gt;</code>.</p>
<i>Nest loops</i>	<p>You cannot nest a <code>&lt;\$_while&gt;</code> in a <code>&lt;\$_while&gt;</code>, or a <code>&lt;\$_repeat&gt;</code> in a <code>&lt;\$_repeat&gt;</code>, in the same macro. Nor can you nest a <code>&lt;\$_while&gt;</code> in an <code>&lt;\$_until&gt;</code>, or an <code>&lt;\$_until&gt;</code> in a <code>&lt;\$_while&gt;</code>. Instead you can call another macro to run a sub-loop. However, you can nest a <code>&lt;\$_while&gt;</code> in a <code>&lt;\$_repeat&gt;</code>, or a <code>&lt;\$_repeat&gt;</code> in a <code>&lt;\$_while&gt;</code>, and you can use one layer of <code>&lt;\$_if&gt;</code> in the mix:</p> <pre>&lt;\$_while (expr)&gt;   &lt;\$_if (expr)&gt;     &lt;\$_repeat (expr)&gt;...&lt;\$_endrepeat&gt;   &lt;\$_else&gt;     &lt;\$_repeat (expr)&gt;...&lt;\$_endrepeat&gt;   &lt;\$_endif&gt; &lt;\$_endwhile&gt;</pre>
<i>Move around within loops</i>	<p>You can use <code>&lt;\$_break&gt;</code> to skip to the end of the loop, and <code>&lt;\$_continue&gt;</code> to jump back to the start of the next iteration. Although you can invoke these control elements in <code>&lt;\$_if&gt;</code>s, it is simpler to use the following constructs:</p> <pre>&lt;\$_break if (expr)&gt; &lt;\$_continue if (expr)&gt;</pre> <p>Use only a single space before each <code>if</code>, and a minimum of one space after each <code>if</code>. These constructs work even in nested <code>&lt;\$_while&gt;</code> or <code>&lt;\$_repeat&gt;</code> loops, where they apply to the innermost of the loops where they occur.</p>

### 37.6.5 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 37-9](#) lists several of the string operators and shows how they are used in macro expressions.

See also:

[§37.6.2 Understanding operands and operators](#) on page 701

[Table 37-6 Operators for HTML macro expressions](#) on page 701

**Table 37-9 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 <i>N</i>)</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 <i>N</i>)</code>	String	First <i>N</i> characters of <code>\$\$string</code>
last	<code>(\$\$string last <i>N</i>)</code>	String	Last <i>N</i> characters of <code>\$\$string</code>
before	<code>(\$\$string before \$\$str)</code>	String	Substring that precedes the first (leftmost) occurrence of <code>\$\$str</code> in <code>\$\$string</code>
after	<code>(\$\$string after \$\$str)</code>	String	Substring that follows the first (leftmost) occurrence of <code>\$\$str</code> in <code>\$\$string</code>
starts	<code>(\$\$string starts \$\$str)</code>	Boolean	True if <code>\$\$str</code> is at the start of <code>\$\$string</code>
ends	<code>(\$\$string ends \$\$str)</code>	Boolean	True if <code>\$\$str</code> is at the end of <code>\$\$string</code>
contains	<code>(\$\$string contains \$\$str)</code>	Boolean	True if <code>\$\$str</code> occurs anywhere in <code>\$\$string</code>
trim first	<code>(\$\$string trim first <i>N</i>)</code>	String	All but the first <i>N</i> characters of <code>\$\$string</code>
trim last	<code>(\$\$string trim last <i>N</i>)</code>	String	All but the last <i>N</i> characters of <code>\$\$string</code>
replace with	<code>(\$\$string replace " " with "_")</code>	String	Each instance of first operand is replaced with second operand
upper	<code>(\$\$string upper)</code>	String	<code>\$\$string</code> is all uppercase
lower	<code>(\$\$string lower)</code>	String	<code>\$\$string</code> 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”.

## 37.6.6 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 §37.4 [Using multiple-value list variables](#) on page 695.

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>
```

### 37.6.7 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.

### 37.6.8 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>\
```

## 37.7 Passing a parameter to a macro

You can pass a single parameter to a macro by enclosing the value of the parameter in parentheses; for examples, see the predefined macros for RTF output listed in [Table 37-2](#) on page 684. **DITA2Go** 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")>
```

## 37.8 Debugging macros

By default, **DITA2Go** 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
```

## 37.9 Deploying macros and macro variables

*In this section:*

- §37.9.1 [Understanding where to use macros and macro variables](#) on page 710
- §37.9.2 [Invoking macros at predetermined points in output](#) on page 710
- §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711
- §37.9.4 [Assigning macros to graphics or tables for HTML](#) on page 713
- §37.9.5 [Using HTML Macro PI markers to invoke macros](#) on page 713
- §37.9.6 [Implementing drop-down text with macros](#) on page 713

### 37.9.1 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 PI** 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 §37.3.2 [Assigning values to macro variables](#) on page 688); however, you can use macro variables only within **DITA2Go** macros.

Configuration settings whose values are not themselves macros cannot include macro variables.

### 37.9.2 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 §27.6.2 [Assigning code to \[Inserts\] keywords for splits and extracts](#) on page 535.

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 DITA2Go 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
```

### 37.9.3 Surrounding or replacing text with code or macros

For a better way to surround text with any arbitrary content, you can include `start`, `end`, `before`, and `after` settings in a format configuration file; see §7.4.8 [Assigning content-adding properties to formats](#) on page 118.

To specify code to be invoked before, after, or in place of a paragraph or character format:

- 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.
```
- Assign to the format one of the `Code*` properties listed in [Table 37-10](#). For example:
 

```
[HTMLParaStyles]
PopHead=CodeBefore
```
- 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 §3.4 [Understanding the rules for configuration settings](#) on page 62.

You can also assign a macro to a format in section [StyleLinkSrc], to provide code for the href attribute of HTML links; see §28.2.4 [Specifying link properties with macros](#) on page 548.

**Table 37-10 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	<i>Does not apply to RTF output</i>

\* For HTML conversions, **DITA2Go** 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.

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>
```

<i>Text properties for RTF</i>	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.
<i>Tables for HTML</i>	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.
<i>Entire document for HTML</i>	You could have a DITA 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 §37.3 <a href="#">Using macro variables</a> on page 687) to include specific content based on user entries.

*See also:*

§37.3.5 [Creating macro variables from paragraph content](#) on page 692

### 37.9.4 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 §32.4.2 [Replacing or surrounding a graphic with macro code](#) on page 615.

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 §33.6 [Using macros to control table properties](#) on page 642 and §33.6.1 [Invoking macros around tables](#) on page 642.

You can also specify macros to be invoked inside tables; see §33.6.5 [Specifying row-group, row, and cell attributes with macros](#) on page 644.

### 37.9.5 Using HTML Macro PI markers to invoke macros

You can specify a macro to be invoked via the **HTML Macro** PI marker. Insert a PI 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 HTML macro usage, you can also specify just the macro name in a **HTML Macro** PI marker, without the angle brackets and dollar sign; **DITA2Go** processes the marker text as though it were a **DITA2Go** macro.

### 37.9.6 Implementing drop-down text with macros

The following sections of the **DITA2Go User's Guide** present examples of macros that incorporate JavaScript to dynamically expand and collapse areas of text:

§16.9.7 [Deploying JavaScript code for drop-down sections](#) on page 271

§16.9.8 [Emulating Web Works Publisher drop-down hotspots](#) on page 275

You can modify these macros for your own purposes.

## 37.10 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 **DITA2Go** configuration settings. This section describes the best approach.

Start from the HTML end. Take one of the output .htm files **DITA2Go** generates from your DITA document and use a plain-text editor to modify the HTML code:

1. Look at the code **DITA2Go** 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 **DITA2Go** 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 §37.1.1 [Defining macros](#) on page 679.

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 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** PI markers before and after each instance of adjacent text. The starting PI marker would contain:

```
<$GrInfoBefore>
```

and the ending PI marker would contain:

```
<$GrInfoAfter>
```

You would keep the same macros defined in [Step 3](#), and get the same result.

5. Tell **DITA2Go** 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 **DITA2Go** 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 §37.1.2 [Invoking a macro](#) on page 683 and §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

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 §37.3 [Using macro variables](#) on page 687 and §37.6.4.2 [Using conditional expressions](#) on page 704.



# 38 Working with processing instructions

---

DITA processing instructions provide a way to introduce HTML or RTF code and **DITA2Go** configuration overrides into your document, without affecting the original content or creating format dependencies. Topics include:

- §38.1 [Understanding DITA2Go PI markers](#) on page 717
- §38.2 [Understanding effects of PI markers](#) on page 718
- §38.3 [Adding attributes with PI markers](#) on page 721
- §38.4 [Assigning properties to PI marker types](#) on page 723
- §38.5 [Inserting code with PI markers](#) on page 724

## 38.1 Understanding DITA2Go PI markers

To satisfy the need for a consistent way to specify presentation details for DITA processing, **DITA2Go** supports including any of an extensive set of XML *processing instructions* (PIs) in your DITA source files. See:

<http://www.w3.org/TR/REC-xml/#sec-pi>

*In this section:*

- §38.1.1 [Understanding DITA2Go PI marker syntax](#) on page 717
- §38.1.2 [Including special characters in PI markers](#) on page 718
- §38.1.3 [Deciding when to use PI markers](#) on page 718

### 38.1.1 Understanding DITA2Go PI marker syntax

In **DITA2Go** a DITA processing instruction is called a *PI marker*. A PI marker has the form:

```
<?dtxxx markertype="markertype" markertype2="markertype2" ... ?>
```

where:

<i>xxx</i>	designates the output type
<i>markertype</i>	names the type of instruction or value
<i>markertype2</i>	is the actual instruction or value.

You can include multiple *key="value"* pseudo-attributes or tokens in the same PI marker, separated by spaces. For example:

```
<?dthtm GraphWidth="80" GraphHeight="100" ?>
```

This PI syntax is supported by all XML editors, and is easily extended to specify any instruction.

**DITA2Go** PI markers include the following output types:

<i>dthtm</i>	all HTML/XML outputs
<i>dtrtf</i>	Word output
<i>dtpdf</i>	print outputs, including such as DocBook <i>dbfo</i>
<i>dtall</i>	all outputs

### 38.1.2 Including special characters in PI markers

For quoting, you can use either ' or ". For quotes within the content, use the other:

```
<?dthtm Config="[Attributes]body= onload='prettyPrint()' " ?>
```

You can also just escape whatever quote you used with a backslash within the quoted material:

```
<?dthtm Config="[Attributes]body= onload=\"prettyPrint()\" " ?>
```

PI markers used to insert HTML code or macros, such as **Code** and **HTML Macro** PI markers, must include an extra backslash in content to escape a required backslash, such as in a Windows file path:

```
<?dthtm HTML Macro='<$(($.\\filename.htm after "<body>") before  
" </body>")>' ?>
```

Doubled backslashes come out as singles in macros; see §37.1.1.3 [Escaping special characters in macro definitions](#) on page 680.

PI markers used for hypertext links might include backslashes in the content. In **HyperJump** and **HyperAnchor** PI markers, the backslashes are simply stripped; whatever followed them remains. In **HyperLink** PI markers, the backslashes remain also.

For PI markers of type **HyperLink**, make sure the file name is a URL, starting with `file:///` before an absolute path beginning with a drive letter (for Windows), like this:

```
<?dthtm HyperLink="file:///D:/oh/site/ohframe.htm" ?>
```

### 38.1.3 Deciding when to use PI markers

PI markers are to be used for things that cannot be specified in DITA, but need to be specified for the output type you are producing. They complement the DITA code, and under no conditions alter its meaning.

For example, you can use PI markers to do any of the following:

- Designate split points and extract extents for HTML (see §27.3.1 [Designating split points](#) on page 526).
- Insert WAI markup (see §34 [Generating WAI markup for HTML](#) on page 649).
- Provide ALink entries for Help systems (see §16.6 [Providing related-topic links for Help systems](#) on page 258).
- Identify context-sensitive help targets (see §16.10 [Setting up Context Sensitive Help \(CSH\)](#) on page 277).
- Assign alternate configuration values to individual tables, graphics, paragraphs, or character spans (see §42.2.2 [Overriding settings with configuration PI markers](#) on page 767).
- Assign `outputclass` values to elements where `outputclass` is not allowed by the DITA specification (for example, see §14.9.5 [Mapping indexterms to variant indexes](#) on page 217).

Wherever you can use a rule (such as a configuration setting), do so; where you cannot, use a PI marker.

## 38.2 Understanding effects of PI markers

Each of the PI marker types listed in [Table 38-1](#) produces a predefined effect when **DITA2Go** encounters a marker of that type in your document.

**Table 38-1 PI marker types with predefined effects**

Marker type	Purpose	Effect
<b>ALink</b>	Help-system associative link	Content is the text of an ALink identifier
<b>BorderFormat or BorderType</b>	Format property override	Specifies a border subformat for the element
<b>Branch</b>	Scoping in maps	Names a section of a map as a branch
<b>Break</b>	Presentation	Inserts a line, column, or page break in the output
<b>CellAttr</b>	Table mark-up for HTML and XML	Content is the value of the HTML <code>&lt;td&gt;</code> tag or <code>&lt;th&gt;</code> tag attribute named by <b>Attr</b> , for the enclosing cell
<b>CellClass</b>	Table mark-up for HTML	Content names the CSS <code>class</code> for a table cell
<b>CellGroup</b>	Table mark-up for HTML	Content specifies the group of a header cell
<b>CellID</b>	Table mark-up for HTML	Overrides generated WAI ID attributes for a cell
<b>CellScope</b>	Table mark-up for HTML	Content specifies the scope of a header cell
<b>CellSpan</b>	Table mark-up for HTML	Assigns the <code>span</code> property to a header cell
<b>Code</b>	Insert HTML or RTF code	Content is used as code; macros are expanded
<b>Config</b>	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
<b>ConrefBranch</b>	Map branch processing	Directs a conref to a named map branch
<b>Delete</b>	No standalone purpose	Deletes itself
<b>DITA*</b>	Provide DITA mark-up	See <a href="#">Table 24-2</a> on page 492
<b>DocBook*</b>	Provide DocBook mark-up	See <a href="#">Table 26-1</a> on page 521
<b>EclipseAnchor</b>	Merge Eclipse Help projects	Marks where a secondary TOC should be inserted
<b>EclipseContext</b>	Context-sensitive help	Content is the context ID for an infopop
<b>EclipseLink</b>	Merge Eclipse Help projects	Content includes path to secondary TOC file
<b>ExtCodeEndChar</b>	Include text from external files	Last character of external file to include
<b>ExtCodeEndLine</b>	Include text from external files	Last line of external file to include
<b>ExtCodeFileEnc</b>	Include text from external files	Encoding of external file
<b>ExtCodeFileLen</b>	Include text from external files	Length of external file in characters
<b>ExtCodeStartChar</b>	Include text from external files	First character of external file to include
<b>ExtCodeStartLine</b>	Include text from external files	First line of external file to include
<b>ExtrBottom</b>	Extract files for HTML	Content is the last item in the <code>&lt;body&gt;</code> of an extract
<b>ExtrDisable</b>	Extract files for HTML	Turns off extract processing
<b>ExtrEnable</b>	Extract files for HTML	Turns on extract processing
<b>ExtrEnd</b>	Extract files for HTML	Ends an extract
<b>ExtrFinish</b>	Extract files for HTML	Marks the last paragraph of an extract
<b>ExtrHead</b>	Extract files for HTML	Content is placed in the <code>&lt;head&gt;</code> of an extract
<b>ExtrReplace</b>	Extract files for HTML	Content replaces an extract in the original file
<b>ExtrStart</b>	Extract files for HTML	Marks the first paragraph of an extract
<b>ExtrTop</b>	Extract files for HTML	Content is the first item in the <code>&lt;body&gt;</code> of an extract
<b>FileName</b>	Split or extract HTML files	Content is the name of a split or extract file

**Table 38-1 PI marker types with predefined effects (continued)**

Marker type	Purpose	Effect
<b>GraphAttr</b>	Image mark-up for HTML and XML	Content is the value of the HTML <code>&lt;img&gt;</code> tag attribute named by <b>Attr</b> for the next image
<b>GraphDpi</b>	Image resolution	Overrides any other DPI setting for the graphic
<b>HelpMerge</b>	Merge Help files	Marks where the named Help file should be inserted
<b>HTMLComment</b>	Add comments to HTML	Content is the text of an HTML comment
<b>HTMConfig</b>	Change configuration setting for HTML	Content is a <code>[Section]Key=Value</code> or <code>[Section]=Value</code> setting for HTML
<b>HyperAlert</b>	Produce a notice	Content is the text of an HTML “alert” notice
<b>HyperAnchor</b>	Provide an anchor for links	Inserts a named <code>&lt;a&gt;</code> tag in HTML
<b>HyperJump</b>	Link to an anchor	Content is the HTML <code>&lt;a href= . . .&gt;</code> attribute
<b>HyperLink</b>	Link to an external URI	Content is the HTML <code>&lt;a href= . . .&gt;</code> attribute
<b>HyperPopup</b>	Pop up a secondary window	Pops up a temporary window in HTML
<b>HyperTarget</b>	Specify a target window	Specifies the window to use for a link destination
<b>JH2PopProp</b>	JavaHelp window property	Content is a JavaHelp 2 pop-up window parameter
<b>JH2SecProp</b>	JavaHelp window property	Content is a JavaHelp 2 secondary window parameter
<b>KeyrefBranch</b>	Map branch processing	Names the map branch to use to resolve the next keyref
<b>LinkAttr</b>	Link mark-up for HTML and XML	Content is the value of the HTML <code>&lt;a href= . . .&gt;</code> tag attribute named by <b>Attr</b> , for the next link
<b>LinkClass</b>	Link mark-up for HTML	Content names the CSS <code>class</code> for the next link
<b>LocalTOCTitle</b>	Split files for HTML	Content is the text of a local-TOC link
<b>MetaType</b>	<code>&lt;meta&gt;</code> tag for HTML	Content is the <code>content</code> value for a new <code>&lt;meta name=Type content=...&gt;</code> tag
<b>Print</b>	Conditional output	Content determines whether current topic is for print output, only for print output, or not for print output
<b>RowAttr</b>	Table mark-up for HTML and XML	Content is the value of the <code>&lt;tr&gt;</code> or <code>&lt;row&gt;</code> attribute named by <b>Attr</b> , for the row of the enclosing cell
<b>RTFConfig</b>	Change configuration setting for RTF	Content is a <code>[Section]Key=Value</code> or <code>[Section]=Value</code> setting for RTF
<b>Search</b>	Conditional output	Content determines whether content is included in FTS
<b>ShadeFormat or ShadeType</b>	Format property override	Specifies a shading subformat for the element
<b>SimpleTableRelCol</b>	Table properties	Specifies relative or absolute column widths for tables
<b>SimpleTableWidth</b>	Table properties	Specifies the absolute width of a table
<b>Split</b>	Split files for HTML	Marks a split point in a DITA file
<b>TableAttr</b>	Table mark-up for HTML and XML	Content is the value of the HTML <code>&lt;table&gt;</code> tag attribute named by <b>Attr</b> , for the enclosing table
<b>Title</b>	Split or extract files for HTML	Content is the page title of a split or extract file
<b>TopicAlias</b>	Context-sensitive help	Inserts a named CSH target in output
<b>TopicStartCode</b>	<code>&lt;head&gt;</code> code for HTML	Code is executed before topic content is processed
<b>Window</b>	HTML Help secondary window	Content names a window for jumps from contents or index
<b>XrefBranch</b>	Map branch processing	Directs a cross reference to a named map branch

You can invent your own custom PI marker types. Any PI marker with a qualifying target (`dtrtf`, `dthtm`, or `dtall`), where **DITA2Go** does not reserve the attribute name, becomes a valid PI marker. The type is the attribute name, and the content is the attribute value. For example:

```
<?dthtm zzTest="My test content" ?>
```

is a PI marker of type `zzTest`, and can be mapped in `[MarkerTypes]`, so you could assign it other PI marker type properties:

```
[MarkerTypes]
zzTest = TopicStartCode
```

When you invent a new custom PI marker type, the name must not conflict with your use of any of the predefined PI marker types listed in [Table 38-1](#).

Avoid adding a custom PI marker type whose name has any of the following characteristics:

- Duplicates the name of a PI marker type listed in [Table 38-1](#), if you intend to use PI markers of that type for the purpose shown.
- 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 PI markers are assumed by **DITA2Go** to be attribute markers; see §38.3 [Adding attributes with PI markers](#) on page 721.
- 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 §20.8.1.5 [Overriding window-access properties with markers](#) on page 407.

## 38.3 Adding attributes with PI markers

An *attribute PI marker* includes the name of the attribute as a suffix to the predefined PI 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** PI marker with content `yellow`, placed just before a DITA `<entry>` tag, would add the attribute `bgcolor` with value `yellow` to the HTML `<tr>` tag for the current table row:

```
<?dthtm Rowbgcolor="yellow" ?>
```

results in:

```
<tr bgcolor="yellow">
```

### Nonconforming attribute markers

A few attribute PI markers do not conform exactly to this naming and usage convention; for example, WAI support PI markers **CellGroup** and **CellSpan**. See §35.2.4 [Assigning table-cell attribute values with PI markers](#) on page 666. 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 DITA PI marker is not limited in length, you can concatenate all PI markers for the same attribute that are inserted before the next item to which they apply. You can just add more PI markers of the same type, and continue the content. For example:

```
<?dthtm tablesummary="This table shows the properties" ?>
<?dthtm tablesummary=" you can use for thingamabobs." ?>
```

Also:



- Inserting another PI marker of a different type between two PI markers for the same attribute does not prevent concatenation, even if the middle PI marker is a different attribute PI marker for the same element.
- If you want the content of two concatenated attribute PI 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 PI marker or at the beginning of content in the second PI marker.

**Extra attributes** Using PI markers to add attributes can result in extra attributes for a given tag. Browsers ignore extra attributes, but validators would not be pleased; see §22.14 [Passing W3C validation tests](#) on page 445. (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 PI markers with identical names but different content apply to the same element, **DITA2Go** uses the content of the last PI marker encountered as the value of the attribute.

For HTML or XML output, **DITA2Go** treats any PI marker that has a name that begins with **Cell**, **Char**, **Graph**, **Link**, **Meta**, **Para**, **Row**, or **Table** as an attribute PI marker. For HTML (for example), **DITA2Go** 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 `<img>` tag.
- LinkAttr** In the next link (`<a href=...>`) tag.
- Meta Type** 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 38-2](#) lists the elements to which each attribute PI marker can apply for each output type.

**Table 38-2 Elements to which attribute PI markers apply, by output type**

Marker	Output type			
	HTML/XHTML	Generic XML	DITA XML	DocBook XML
<b>CellAttr</b>	<code>&lt;td&gt;</code> , <code>&lt;th&gt;</code>	<code>&lt;td&gt;</code> , <code>&lt;th&gt;</code>	<code>&lt;entry&gt;</code> , <code>&lt;stentry&gt;</code> , <code>&lt;choption&gt;</code> , <code>&lt;chdesc&gt;</code> , <code>&lt;proptype&gt;</code> , <code>&lt;propvalue&gt;</code> , <code>&lt;propdesc&gt;</code>	<code>&lt;td&gt;</code> , <code>&lt;th&gt;</code>
<b>CharAttr</b>	<i>inline elements</i>	<i>inline elements</i>	<i>inline elements</i>	<i>inline elements</i>
<b>GraphAttr</b>	<code>&lt;img&gt;</code>	<code>&lt;img&gt;</code>	<code>&lt;image&gt;</code>	<code>&lt;imagedata&gt;</code>
<b>LinkAttr</b>	<code>&lt;a&gt;</code> ( (does not apply to Help pop-ups, secondary window jumps, or footnote cross references)	<code>&lt;a&gt;</code> (applies to the <code>AtagElement</code> setting; do not use for name; overridden by <code>XMLLinkAttrs</code> )	<code>&lt;xref&gt;</code> (can add to or replace standard <code>href</code> , <code>type</code> , <code>format</code> , and <code>scope</code> attributes)	<code>&lt;xref&gt;</code> , <code>&lt;ulink&gt;</code>

**Table 38-2 Elements to which attribute PI markers apply, by output type**

Marker	Output type			
	HTML/XHTML	Generic XML	DITA XML	DocBook XML
<b>MetaType</b>	<meta>	<meta>		
<b>ParaAttr</b>	<i>block elements</i>	<i>block elements</i>	<i>block elements</i>	<i>block elements</i>
<b>RowAttr</b>	<tr>	<tr>	<row>, <strow>, <chrow>, <property>	<tr>, <row>
<b>TableAttr</b>	<table>	<table>	<table>, <simplatable>, <choicetable>, <properties>	<table>

## 38.4 Assigning properties to PI marker types

You can define the behavior of a new PI marker type, or redefine the behavior of an existing PI marker type, by assigning one or more properties to the PI marker type:

```
[MarkerTypes]
; PI marker type name = properties
Marker = Property1 Property2 ...
```

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 38-3](#) lists all the `[MarkerTypes]` properties, shows which output types apply, and describes the effect of each property.

**Table 38-3 Effects of `[MarkerTypes]` properties**

Output	Property	Effect
RTF or HTML	<code>ALink</code>	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>Code</code>	Any macros in the marker are expanded, and the content is surrounded by any code specified for the marker type in <code>[MarkerTypeCodeBefore]</code> and <code>[MarkerTypeCodeAfter]</code> ; see §38.5 <a href="#">Inserting code with PI markers</a> on page 724 for more information. <i>Cannot be combined with <code>HTMLComment</code>.</i>
	<code>Config</code>	Content is a configuration setting of the form <code>[Section]Key=Value</code> or <code>[Section]=Value</code> ; see §42.2 <a href="#">Overriding settings with PI markers or macros</a> on page 766.
	<code>Delete</code>	The marker is removed entirely; <b>DITA2Go</b> applies this property last, after any other properties you specify. <i>Must be specified last.</i>
RTF only	<code>RTFConfig</code>	Content is a configuration setting of the form <code>[Section]Key=Value</code> or <code>[Section]=Value</code> ; see §42.2 <a href="#">Overriding settings with PI markers or macros</a> on page 766.
HTML only	<code>ANSI</code>	Specifies the Windows code page to use, default 1252; or 1250 for CE/EE, 1251 for Cyrillic, 1253 for Greek, 1254 for Turkish.
	<code>Extr*</code>	Each of these markers has the same effect as the corresponding <code>[HTMLParaStyles]parafmt=Extr*</code> property; see §27.4.1 <a href="#">Enabling and disabling extract processing</a> on page 528. For <code>ExtrDisable</code> , <code>ExtrEnable</code> , <code>ExtrEnd</code> , and <code>ExtrStart</code> , any content is ignored, unless you also specify other properties that use the content, such as <code>Code</code> .
	<code>ExtrBottom</code>	Content becomes the last item in the extract <code>&lt;body&gt;</code> .
	<code>ExtrDisable</code>	Turns extract processing off.
	<code>ExtrEnable</code>	Turns extract processing on.

**Table 38-3 Effects of [MarkerTypes] properties (continued)**

Output	Property	Effect
	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 §43.3.2 <a href="#">Using PI markers to name output files</a> on page 782)
	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 §42.2 <a href="#">Overriding settings with PI markers or macros</a> on page 766.
	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 DITA file; has the same effect as [HTMLParaStyles]parafmt=Split; see §27.3.1 <a href="#">Designating split points</a> on page 526. 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>
	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.

## 38.5 Inserting code with PI markers

**DITA2Go** can insert the content of a PI marker directly in output, at the location where it occurs in your DITA document. This is what happens by default to the content of any PI marker of type **Code**. For example, you could include a literal `&nbsp;` in your HTML output with:

```
<?dthtml code="&nbsp;" ?>
```

If **Code** PI marker content includes **DITA2Go** macros, the macros are expanded.

To place PI marker content at a location in HTML or XML that is outside of any paragraph, dedicate a paragraph format to this use (for example, *MarkerOnly*). Put the PI marker in an otherwise empty element to which you have assigned the *MarkerOnly* paragraph format, and assign the following property:

```
[HTMLParaStyles]
MarkerOnly = Raw
```

See §30.2.4 [Stripping paragraph properties](#) on page 568.

If you assign the Code property to a PI marker type (see §38.4 [Assigning properties to PI marker types](#) on page 723), you can have **DITA2Go** surround the content with additional “before” and “after” code, or replace the content with code:

```
[MarkerTypeCodeBefore]
; marker type name = macro
; for PI markers assigned the Code property in MarkerTypes.
```

```
[MarkerTypeCodeAfter]
; marker type name = macro
; for PI markers assigned the Code property in MarkerTypes.

[MarkerTypeCodeReplace]
; marker type name = macro
; for PI markers assigned the Code property in MarkerTypes.
```

**DITA2Go** expands macros assigned in these sections. If the PI marker type is assigned the Code property and PI marker content includes macros, those macros are expanded, also.

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, **DITA2Go** 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 PI 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 PI 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 **DITA2Go** 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 **DITA2Go** is ready to write the <head> of the topic, the <\$\_repeat> loop (see §37.6.4.3 [Using loop structures](#) on page 705) 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.

# 39 Working with templates

---

This section explains how to use **DITA2Go** configuration templates. Topics include:

- §39.1 [Working with configuration templates](#) on page 727
- §39.2 [Referencing configuration files and templates](#) on page 731
- §39.3 [Including document-specific configuration files](#) on page 732
- §39.4 [Deciding which configuration file to edit](#) on page 734
- §39.5 [Creating your own configuration templates](#) on page 741

## 39.1 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. **DITA2Go** 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:*

- §39.1.1 [Understanding how templates are organized](#) on page 727
- §39.1.2 [Understanding how templates are named](#) on page 728
- §39.1.3 [Understanding how templates are chained together](#) on page 728
- §39.1.4 [Understanding how macro libraries are organized](#) on page 729
- §39.1.5 [Understanding how format templates are organized](#) on page 730
- §39.1.6 [Understanding how language templates are organized](#) on page 730

### 39.1.1 Understanding how templates are organized

Your **DITA2Go** distribution includes templates that are linked together by references that extend through the chains, from your project configuration file through templates in %OMSYSHOME%\d2g subdirectories, to the very end of the configuration chain at %OMSYSHOME%\common\system\config\omsys.ini. See §39.1.3 [Understanding how templates are chained together](#) on page 728

Configuration templates include the following groups:

- [General configuration settings](#), located in `config` directories
- [Macro definitions](#), located in `macros` directories
- [Output format definitions](#), located in `formats` directories
- [Output text settings by language](#), located in `lang` 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:

	<code>%OMSYSHOME%\d2g\system\config: default general configuration settings</code>
	<code>%OMSYSHOME%\d2g\local\config: your overrides to the system settings</code>
<b>Macro definitions</b>	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: <code>%OMSYSHOME%\d2g\system\macros: definitions of general-purpose macros</code> <code>%OMSYSHOME%\d2g\local\macros: definitions of your own macros</code>
<b>Output format definitions</b>	The presentation styles of paragraphs, characters, tables, and page layouts are governed by format configuration templates. These templates provide an easy way to define how you want the output to look. Format configuration templates are located in the following directory: <code>%OMSYSHOME%\d2g\system\formats: default format definitions and page layouts</code> <code>%OMSYSHOME%\d2g\local\formats: your format definitions and page layouts</code>
<b>Output text settings by language</b>	<b>DITA2Go</b> generates several headings and labels that appear as text in your output. Each language configuration template includes text for these headings and labels in one language. Language configuration templates are located in the following directories: <code>%OMSYSHOME%\d2g\system\lang: text for generated headings and labels</code> <code>%OMSYSHOME%\d2g\local\lang: your overrides to headings and labels</code>

### 39.1.2 Understanding how templates are named

Template files in your **DITA2Go** 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, formats, lang, or macro):

<u>Template name</u>	<u>Editable file name</u>	<u>Scope of settings</u>
<code>omsys.ini</code>	<code>local_omsys.ini</code>	All Omni Systems projects
<code>d2g_type.ini</code>	<code>local_d2g_type.ini</code>	All <b>DITA2Go</b> projects
<code>d2htm_type.ini</code>	<code>local_d2htm_type.ini</code>	<b>DITA2Go</b> HTML and XML projects
<code>d2rtf_type.ini</code>	<code>local_d2rtf_type.ini</code>	<b>DITA2Go</b> RTF projects

Most general configuration templates, language templates, format templates, and macro libraries use this naming convention.

### 39.1.3 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 39-1 Output-type-specific general configuration files**

<b>Output type</b>	<b>Project configuration file</b>	<b>Editable local configuration file</b>
DITA	<code>_d2dita.ini</code>	<code>local_d2dita_config.ini</code>
DocBook	<code>_d2docbook.ini</code>	<code>local_d2docbook_config.ini</code>
Eclipse Help	<code>_d2eclipse.ini</code>	<code>local_d2eclipse_config.ini</code>
HTML	<code>_d2html.ini</code>	<code>local_d2html_config.ini</code>
MS HTML Help	<code>_d2htmlhelp.ini</code>	<code>local_d2htmlhelp_config.ini</code>
JavaHelp	<code>_d2javahelp.ini</code>	<code>local_d2javahelp_config.ini</code>
OmniHelp	<code>_d2omnihelp.ini</code>	<code>local_d2omnihelp_config.ini</code>



**Table 39-1 Output-type-specific general configuration files (continued)**

Output type	Project configuration file	Editable local configuration file
Oracle Help	_d2oraclehelp.ini	local_d2oraclehelp_config.ini
WinHelp	_d2winhelp.ini	local_d2winhelp_config.ini
MS Word	_d2rtf.ini	local_d2rtf_config.ini
XHTML	_d2xhtml.ini	local_d2xhtml_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 `_d2rtf.ini` references this chain of general configuration templates and files:

```
_d2rtf.ini ->
  local_d2rtf_config.ini-> d2rtf_config.ini ->
    local_d2g_config.ini -> d2g_config.ini ->
      local_omsys.ini -> omsys.ini
```

The HTML starting project configuration file references this chain:

```
_d2html.ini ->
  local_d2htm_config.ini -> d2htm_config.ini ->
    local_d2g_config.ini -> d2g_config.ini ->
      local_omsys.ini -> omsys.ini
```

Some chains are longer. For example, for OmniHelp output, the chain looks like this:

```
_d2omnihelp.ini ->
  local_d2omnihelp_config.ini -> d2omnihelp_config.ini ->
    local_d2help_config.ini -> d2help_config.ini ->
      local_d2htm_config.ini -> d2htm_config.ini ->
        local_d2g_config.ini -> d2g_config.ini ->
          local_omsys.ini -> omsys.ini
```

All general configuration chains go through either `d2htm_config.ini` (for HTML or XML output) or `d2rtf_config.ini` (for Word or WinHelp output). These two configuration templates reference the format and 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:*

§39.1.4 [Understanding how macro libraries are organized](#) on page 729

§39.1.5 [Understanding how format templates are organized](#) on page 730

### 39.1.4 Understanding how macro libraries are organized

Two **DITA2Go** general configuration templates, `d2htm_config.ini` and `d2rtf_config.ini`, respectively reference `local_d2htm_macros.ini` and `local_d2rtf_macros.ini`; and these two editable macro library files in turn reference `d2htm_macros.ini` and `d2rtf_macros.ini`, respectively.

For example, in `d2htm_config.ini`:

```
[Templates]
; Macros = path to macro library
Macros = %OMSYSHOME%\d2g\local\macros\local_d2htm_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 §37.2.4 [Including macro definitions in your own macro library](#) on page 685.

### 39.1.5 Understanding how format templates are organized

Your **DITA2Go** distribution includes the output-format configuration templates and files listed in [Table 39-7](#) on page 739. These files form their own side chain, branching from the general configuration chain of templates and files. To access this chain, the two **DITA2Go** output-category configuration templates, `d2htm_config.ini` and `d2rtf_config.ini`, respectively reference `local_d2htm_formats.ini` and `local_d2rtf_formats.ini`; and the latter in turn reference their template counterparts, then the rest of the format configuration chain.

For example, in `d2rtf_config.ini`:

```
[Templates]
Formats = %OMSYSHOME%\d2g\local\formats\local_d2rtf_formats.ini
```

Because the formats chain is referenced from the **DITA2Go** general configuration chain, you do not have to include a setting for `Formats` in your starting project configuration file, unless you are supplying your own separate formats configuration file(s).

Each of the main format configuration templates references a subformat configuration file and a table format configuration file. For example:

```
[Templates]
Subformats = %OMSYSHOME%\d2g\local\formats\local_d2htm_subformats.ini
Tables = %OMSYSHOME%\d2g\local\formats\local_d2htm_tables.ini
```

The RTF formats configuration template also references a subformat and a page-layout configuration file:

```
[Templates]
Subformats = %OMSYSHOME%\d2g\local\formats\local_d2rtf_subformats.ini
Tables = %OMSYSHOME%\d2g\local\formats\local_d2rtf_tables.ini
Pages = %OMSYSHOME%\d2g\local\formats\local_d2rtf_pages.ini
```

Table, subformat, and page-layout configuration files can be referenced *only* from a formats template, not from a general configuration template, nor from your project configuration file. **DITA2Go** ignores any such out-of-chain references to these files.

See §7.2 [Working with format configuration files](#) on page 110.

### 39.1.6 Understanding how language templates are organized

Your **DITA2Go** distribution includes the language configuration templates and files listed in [Table 39-8](#) on page 740. The top-level **DITA2Go** general configuration template, `d2g_config.ini`, references the editable English language configuration file: `local_d2g_lang_en.ini`:

```
[Templates]
Languages = %OMSYSHOME%\d2g\local\lang\local_d2g_lang_en.ini
```

If you expect to produce output in English, because the English language file is referenced from the **DITA2Go** general configuration chain, you do not have to include a setting for `Languages` in your starting project configuration file. However, to produce output with labels and headings in German (for example), in your project configuration file you would include the following setting:

```
[Templates]
Languages = %OMSYSHOME%\d2g\local\lang\local_d2g_lang_de.ini
```

You can copy any of these language templates and modify the copy to create a regional language variation. Give your new language template a name of the form

d2g\_lang\_LLRR.ini, where:

- LL is the language abbreviation (such as en for English)
- RR is the optional region abbreviation (such as us for United States).

To determine an appropriate abbreviation, see:

<http://www.w3.org/TR/REC-html40/struct/dirlang.html>

<http://www.ietf.org/rfc/rfc1766.txt>

For example, to provide regional differences for the English language, you could prepare the following variants:

local_d2g_lang_enus.ini	United States
local_d2g_lang_enuk.ini	United Kingdom
local_d2g_lang_enoz.ini	Australia
local_d2g_lang_ennz.ini	New Zealand

Your project configuration file could reference one of these variants; each variant should point in turn to d2g\_lang\_en.ini for common values.

It is a good idea to keep your language configuration files in directory

%OMSYSHOME%\d2g\local\lang, and name them as indicated; if an instance of

@xml:lang in your DITA document references a language other than the default,

**DITA2Go** can find the relevant language string.

However, within a referenced language configuration template you can include another [Templates]Languages setting, to make a chain of language configuration templates to be searched; the chain can be any length. All files in the chain must have distinct names; the chain stops if **DITA2Go** finds a repeat. See §39.2 [Referencing configuration files and templates](#) on page 731.

**DITA2Go** resets the language to the default at the start of each topic and map, and also before writing Related Links. For each instance of @xml:lang in your DITA document, **DITA2Go** sets the designated language for that element, and then restores the previous language at the end of the element. If @xml:lang=" ", **DITA2Go** treats this as a request for the default language.

See §8.9 [Localizing output headings, labels, and names](#) on page 157.

## 39.2 Referencing configuration files and templates

To reference a **DITA2Go**-provided general configuration file or template:

```
[Templates]
Configs = %OMSYSHOME%\path\to\sometemplate.ini
```

Your **DITA2Go** distribution includes configuration templates already chained together through references like this; see §39.1.3 [Understanding how templates are chained together](#) on page 728.

When **DITA2Go** 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%\d2g\local\config\local_d2rtf_config.ini
```

If you want to insert another configuration file (for example, `myspecial.ini`) in the chain between the project configuration file and `local_d2rtf_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 §39.5 [Creating your own configuration templates](#) on page 741.

Settings that specify paths to configuration templates, or to any other files in the **DITA2Go** distribution directory structure, should use absolute paths that begin with environment variable `%OMSYSHOME%`; for example:

```
Configs = %omsysHOME%\d2g\local\config\local_d2htm_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 **DITA2Go** 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.

See §42.1.2 [Understanding precedence of configuration settings](#) on page 765.

## 39.3 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 DITA document (though possibly to multiple output types from that document).

*In this section:*

§39.3.1 [Referencing a document-specific configuration file](#) on page 732

§39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733

§39.3.3 [Indicating the intended scope of a configuration file](#) on page 734

### 39.3.1 Referencing a document-specific configuration file

To reference a document-specific configuration file:

```
[Templates]
; Document = path to document-specific configuration file
Document = %OMSYSHOME%\d2g\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%\d2g\documents directory included in your **DITA2Go** distribution for this purpose) or in a `_config` subdirectory under your DITA source directory; see §39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733.

**DITA2Go** 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
<code>Formats</code>	Output formats
<code>Lang</code>	Text of headings and labels

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.

**Table 39-2 Configuration options determined at run time**

Configuration section	Option	Ref.
Setup	FileSuffix	<a href="#">4.1.6</a>
HelpOptions	MakeCombinedCnt	<a href="#">17.2.6</a>

### 39.3.2 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 DITA 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.](#)

Regardless of which default you choose, you will be able to specify a different choice for each project, when you configure default settings; see §2.3 [Configuring default DITA2Go project settings](#) on page 40.

*Single output type  
per document*

If you expect to produce only one output type from each DITA 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 <b>Local</b> ; configuration files specific to a given document will be stored in a directory adjacent to the output directory for each project.
<i>Multiple output types per document</i>	If you expect to produce more than one output type from each DITA document, the best place is <b>Local</b> ; configuration files specific to a given document will be stored in a directory adjacent to the output directory for each project.
<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 <b>Central</b> , and use the <i>Browse</i> dialog to specify a parent directory for document-specific subdirectories. The default parent directory is %OMSYSHOME%\d2g\document. You select or create a subdirectory for each source document, at the time you create a conversion project for that 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 <b>Central</b> .
<i>Lone writer</i>	If you are the only person working on your projects, <b>Central</b> 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 <b>Local</b> ; configuration files specific to a given document will be stored in a subdirectory adjacent to the output directory for each project.

### 39.3.3 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 **DITA2Go** distribution, that value overrides any internal *Scope* value maintained by the Configuration Manager.

## 39.4 Deciding which configuration file to edit

Each **DITA2Go** 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:*

- §39.4.1 [Understanding what configuration files are available](#) on page 735
- §39.4.2 [Editing a project configuration file](#) on page 737
- §39.4.3 [Editing a document-specific configuration file](#) on page 738
- §39.4.4 [Editing an output-specific configuration file](#) on page 738
- §39.4.5 [Editing a format configuration file](#) on page 739
- §39.4.6 [Editing a language configuration file](#) on page 740
- §39.4.7 [Editing a macro configuration file](#) on page 740



§39.4.8 [Indicating the intended scope of a configuration file](#) on page 741

See also:

§1.3.7 [Establish system-wide configuration settings](#) on page 33

§42.1 [Using a different configuration for selected files](#) on page 765

### 39.4.1 Understanding what configuration files are available

Most of the supplied **DITA2Go** configuration files available for editing are located in subdirectories of %OMSYSHOME%\d2g\local, with the following exceptions:

- Starting project configuration file, copied by **DITA2Go** the **DITA2Go** Project Manager from %OMSYSHOME%\d2g\local\starts (or from %OMSYSHOME%\d2g\system\starts) to your project directory
- Document-specific configuration files in the document \_config subdirectory; see §39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733
- Specialization-supporting configuration files in subdirectories of %OMSYSHOME%\d2g\specializations
- Site-specific configuration file local\_omsys.ini, located in %OMSYSHOME%\common\local\config.

[Table 39-3](#) lists the types of configuration files, shows where the files are located, and indicates the intended scope of settings for each type.

**Table 39-3 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	d2g\documents or ..\_config (parallel to project directory)	All or most outputs to be generated from a given DITA source document
	Output-specific	d2g\local\config	All or most DITA source documents to be converted to a given output type
	Specialization-specific	d2g\specializations\..	All DITA sources that include the specialization
	<b>DITA2Go-wide</b>	d2g\local\config	All <b>DITA2Go</b> projects for all documents and outputs
	Site-wide	common\local\config	All Omni Systems projects
Format	Output category	d2g\local\formats	All or most DITA source documents to be converted to a set of output types
Language	Usually universal	d2g\local\lang	All or most DITA source documents
Macro	Usually output-specific	d2g\local\macros	All or most outputs of a given output type or set of output types

Each configuration file in a \d2g\local subdirectory references an eponymous configuration template in a \d2g\system subdirectory that contains default settings. *Do not edit those referenced templates*, because they will be overwritten whenever you update **DITA2Go**. Instead, override settings in the corresponding \local configuration file.

[Table 39-4](#) 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 39-4](#) 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 39-4](#). 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, **DITA2Go** 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 39-4 Chain of general configuration files for HTML Help output**

Scope of settings		General configuration file
All Omni Systems projects	System:	%OMSYSHOME%\common\system\config\omsys.ini
	Local site:	%OMSYSHOME%\common\local\config\local_omsys.ini
All <b>DITA2Go</b> projects	System:	%OMSYSHOME%\d2g\system\config\d2g_config.ini
	Local site:	%OMSYSHOME%\d2g\local\config\local_d2g_config.ini
All HTML/XML projects	System:	%OMSYSHOME%\d2g\system\config\d2htm_config.ini
	Local site:	%OMSYSHOME%\d2g\local\config\local_d2htm_config.ini
All Help projects	System:	%OMSYSHOME%\d2g\system\config\d2help_config.ini
	Local site:	%OMSYSHOME%\d2g\local\config\local_d2help_config.ini
All HTML Help projects	System:	%OMSYSHOME%\d2g\system\config\d2htmlhelp_config.ini
	Local site:	%OMSYSHOME%\d2g\local\config\local_d2htmlhelp_config.ini
All output types from this source document		..\_config\docname_document.ini (under source directory) -- or -- %OMSYSHOME%\d2g\documents\docname_document.ini
This project only		_d2htmlhelp.ini (in project directory)

*General  
configuration  
settings*

If you have just one DITA document to convert to a single output type, most general configuration settings can go in the project configuration file; see §39.4.2 [Editing a project configuration file](#) on page 737.

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 §39.4.3 [Editing a document-specific configuration file](#) on page 738.

If you have many DITA 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 §39.4.4 [Editing an output-specific configuration file](#) on page 738.

*Format  
configuration files*

To override **DITA2Go** default format configuration settings, redefine output formats by adding the definitions you prefer to the appropriate output-oriented format configuration files in d2g\local\formats. See §39.4.5 [Editing a format configuration file](#) on page 739.

*Language  
configuration files*

To override **DITA2Go** default text values for the output language you wish to use, specify settings for alternate text in the appropriate language configuration file in d2g\local\lang. See §39.4.6 [Editing a language configuration file](#) on page 740.

*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 d2g\local\macros. See §39.4.7 [Editing a macro configuration file](#) on page 740.

## 39.4.2 Editing a project configuration file

**DITA2Go** maintains a set of annotated templates for starting project configuration files, in directory %OMSYSHOME%\d2g\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 **DITA2Go**. Instead, if you want to customize a starting project configuration file for your particular operating environment, copy the appropriate file to %OMSYSHOME%\d2g\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 set up a new conversion project, the **DITA2Go** Project Manager copies a starting project configuration file, named for the output type you specified, into your project directory. 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 **DITA2Go** Project Manager copies to the project directory. [Table 39-5](#) lists the names of these files.

**Table 39-5 Output types and starting project configuration files**

Category	Output type	Project configuration file	Ref.
HTML-based Help	Eclipse Help	_d2eclipse.ini	<a href="#">21</a>
	Microsoft HTML Help	_d2htmlhelp.ini	<a href="#">18</a>
	JavaHelp	_d2javahelp.ini	<a href="#">20</a>
	OmniHelp	_d2omnihelp.ini	<a href="#">19</a>
	Oracle Help for Java	_d2oraclehelp.ini	<a href="#">20</a>
HTML	Standard HTML 4.0	_d2html.ini	<a href="#">22</a>
	XHTML 1.0	_d2xhtml.ini	<a href="#">22</a>
XML	DITA XML	_d2dita.ini	<a href="#">24</a>
	Docbook XML	_d2docbook.ini	<a href="#">26</a>
	Generic XML	_d2xml.ini	<a href="#">23</a>
RTF	WinHelp	_d2winhelp.ini	<a href="#">17</a>
	Print RTF	_d2rtf.ini	<a href="#">15</a>

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%\d2g\local\config\local_d2rtf_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 §39.4.3 [Editing a document-specific configuration file](#) on page 738 and §39.4.4 [Editing an output-specific configuration file](#) on page 738.

The **DITA2Go** Project Manager helps you specify additional settings for both starting and source-specific configurations.

### 39.4.3 Editing a document-specific configuration file

For settings that are specific to your DITA source document and that apply to a group of output types, **DITA2Go** maintains one or more document-specific configuration files. By default, document-specific configuration files for all your DITA documents are located in directory %OMSYSHOME%\d2g\documents. However, you can choose to keep such files elsewhere; see §39.3.2 [Deciding where to keep document-specific configuration files](#) on page 733.

When you set up a new project, **DITA2Go** 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 **DITA2Go** project manager sets up the document-specific configuration file for you the first time you create a conversion project for a particular document. This file is named for the map file you specify when you create the project. Later conversion projects for the same document use the same source-specific configuration file.

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 §39.3 [Including document-specific configuration files](#) on page 732.

### 39.4.4 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 DITA source documents, **DITA2Go** maintains user-modifiable output-specific general configuration files in directory %OMSYSHOME%\d2g\local\config. You can customize these files with settings that are appropriate for your particular environment. [Table 39-6](#) lists the output-specific general configuration files you can edit.

**Table 39-6 Editable local output-specific configuration files**

Category	Output type	Editable configuration file	Ref.
HTML-based Help	<i>All HTML-based Help outputs</i>	local_d2help_config.ini	<a href="#">16</a>
	Eclipse Help	local_d2eclipse31_config.ini local_d2eclipse33_config.ini	<a href="#">21</a>
	Microsoft HTML Help	local_d2htmlhelp_config.ini	<a href="#">18</a>
	JavaHelp	local_d2javahelp_config.ini	<a href="#">20</a>
	OmniHelp	local_d2omnihelp_config.ini	<a href="#">19</a>
	Oracle Help for Java	local_d2oraclehelp_config.ini	<a href="#">20</a>
HTML	<i>All HTML-based outputs</i>	local_d2html_config.ini	<a href="#">22</a>
	XHTML 1.0	local_d2xhtml_config.ini	<a href="#">22</a>

**Table 39-6 Editable local output-specific configuration files (continued)**

Category	Output type	Editable configuration file	Ref.
XML	DITA XML	local_d2dita_config.ini	<a href="#">24</a>
	Docbook XML	local_d2docbook_config.ini	<a href="#">26</a>
	Generic XML	local_d2xml_config.ini	<a href="#">23</a>
RTF	WinHelp	local_d2winhelp_config.ini	<a href="#">17</a>
	<i>All RTF-based outputs</i>	local_d2rtf_config.ini	<a href="#">15</a>

Near the top of each configuration file you will find a setting that links that file to an eponymous configuration template in `\d2g\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 **DITA2Go**. Instead, override settings as needed in the editable configuration files.

For example, in `local_d2xhtml_config.ini`:

```
[Templates]
; Where the rest of the configuration settings are:
Configs=%omsyshome%\d2g\system\config\d2xhtml_config.ini
```

Avoid disturbing this setting, or you might break the chain that leads to all the other configuration settings for your conversion project.

### 39.4.5 Editing a format configuration file

For settings that define output formats, **DITA2Go** maintains user-modifiable format configuration files in directory `%OMSYSHOME%\d2g\local\formats`. You can customize these files with settings that are appropriate to your particular needs. [Table 39-7](#) lists the format configuration files you can modify.

**Table 39-7 Output types and format configuration files**

Output type	Editable format configuration file
HTML	local_d2htm_formats.ini
	local_d2htm_subformats.ini
	local_d2htm_tables.ini
	local_d2oraclehelp_tables.ini
RTF	local_d2rtf_formats.ini
	local_d2rtf_subformats.ini
	local_d2rtf_tables.ini
	local_d2rtf_pages.ini
	local_d2winhelp_formats.ini
	local_d2winhelp_tables.ini

The main format configuration files (those with `_formats` in their names) are referenced from output-specific general configuration files. The main format files reference subformat and table configuration files, and for Word output, a page-layout configuration file.

Near the top of each editable format configuration file you will find a setting that links that file to an eponymous configuration template in a `\d2g\system\formats` 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 **DITA2Go**. Instead, override settings as needed in the editable format configuration files in `\d2g\local\formats`.

Each format configuration file contains settings commonly needed for the output type indicated by the name of the file. Near the top of each format configuration file you will find a setting that links that file to other format configuration files. For example, in `local_d2rtf_formats.ini`:

```
[Templates]
; Where the rest of the format configuration settings are:
Formats=%omsyshome%\d2g\system\formats\d2rtf_formats.ini
```

Avoid disturbing this setting, or you might break the chain that connects to all the other format settings for your conversion project.

Before you add settings to the editable format configuration files, see:

§7 [Configuring output formats](#) on page 109

§8 [Configuring format components](#) on page 141.

### 39.4.6 Editing a language configuration file

**DITA2Go** provides a collection of language configuration files that contain settings for the text of various output headings, labels, and names, in different languages. These files are located in `%OMSYSHOME%\d2g\local\lang`. You can customize these files with values that are appropriate to your particular needs. [Table 39-8](#) lists the language configuration files you can edit.

**Table 39-8 Language configuration files**

Language	Editable language configuration file
Czech	<code>local_d2g_lang_cz.ini</code>
German	<code>local_d2g_lang_de.ini</code>
English	<code>local_d2g_lang_en.ini</code>
Spanish	<code>local_d2g_lang_es.ini</code>
French	<code>local_d2g_lang_fr.ini</code>
Russian	<code>local_d2g_lang_ru.ini</code>

The English language configuration file is referenced by the configuration template at the top of the general-configuration chain: `%OMSYSHOME%\d2g\system\d2g_config.ini`:

```
[Templates]
Languages=%omsyshome%\d2g\local\lang\local_d2g_lang_en.ini
```

If you plan to produce output in a language other than English, you can include a reference in your starting configuration file to point to a different language-specific template. For example, to reference the German version, add this setting:

```
[Templates]
; Languages = path to main language configuration template
Languages = %OMSYSHOME%\d2g\local\lang\local_d2g_lang_de.ini
```

This setting will override the default setting in `d2g_config.ini`.

See §8.9 [Localizing output headings, labels, and names](#) on page 157.

### 39.4.7 Editing a macro configuration file

**DITA2Go** provides several macro libraries in the form of macro configuration files, organized by output type. These files are located in `%OMSYSHOME%\d2g\local\macros`.

You can add your own macros, and override macros in the default macro libraries they reference. [Table 39-9](#) lists the macro configuration files you can edit.

**Table 39-9 Macro configuration files**

Output type	Editable macro configuration file
HTML	local_d2g_d2htm_macros.ini
Print RTF	local_d2g_d2rtf_macros.ini
WinHelp	local_d2g_d2winhelp_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 **DITA2Go**. *Do not edit those referenced libraries*; they will be overwritten every time you update **DITA2Go**. To change a distributed macro, override it with a new definition in the appropriate `\local\macros` library.

### 39.4.8 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 **DITA2Go** distribution, that value overrides any internal `Scope` value maintained for that file by the Configuration Manager.

## 39.5 Creating your own configuration templates

Besides using the configuration templates supplied with your **DITA2Go** 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:*

- §39.5.1 [Creating a template from a project configuration file](#) on page 741
- §39.5.2 [Deciding what to include in a general configuration template](#) on page 742
- §39.5.3 [Chaining configuration templates](#) on page 743

*See also:*

- §3.1 [Working with DITA2Go configuration files](#) on page 49
- §39.2 [Referencing configuration files and templates](#) on page 731
- §42.1 [Using a different configuration for selected files](#) on page 765

### 39.5.1 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 **DITA2Go**; 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 §39.5.2 [Deciding what to include in a general configuration template](#) on page 742.

### 39.5.2 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.

**DITA2Go** 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 **DITA2Go**-supplied file in the chain. However, you might prefer to add settings to the appropriate editable configuration file supplied in the existing chain; see §39.4 [Deciding which configuration file to edit](#) on page 734.

#### *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 §42.1.2 [Understanding precedence of configuration settings](#) on page 765.

#### *Define macros elsewhere*

Do not include macro definitions in a general configuration template; keep macro definitions in a separate library file; see §37.2.4 [Including macro definitions in your own macro library](#) on page 685 and §39.1.4 [Understanding how macro libraries are organized](#) on page 729.

#### *Define output formats elsewhere*

Do not include output format definitions in a general configuration template; those must be added to a template specifically for formats, and referenced via one of the following settings:

```
[Templates]
Formats = path/to/myformats.ini
Tables = path/to/mytables.ini
; For RTF output only:
Pages = path/to/mypages.ini
```

See §7.2 [Working with format configuration files](#) on page 110 and §39.1.5 [Understanding how format templates are organized](#) on page 730.

#### *Some settings are duplicated*

Although the settings in [Table 39-2](#) can be included in a configuration template, some will end up in the project configuration file anyway; either because **DITA2Go** originates them, or because their values can be changed at run time. If you remove one of these settings



from the project configuration file, **DITA2Go** will put it back in, at the end of the section where it belongs. If the section itself is missing, **DITA2Go** places the section and the setting near the end of the project configuration file.

### 39.5.3 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 **DITA2Go** to search for settings. The chain can be any length. All files in the chain must have distinct names; the search stops if **DITA2Go** 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.

#### *Precedence of templates*

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 §42.1.2 [Understanding precedence of configuration settings](#) on page 765.



# 40 Working with graphics

---

This section tells how to manage the graphics referenced by your DITA document, and control their appearance in the output produced by **DITA2Go**. Topics include:

§40.1 [Choosing an appropriate graphics format](#) on page 745

§40.2 [Replacing and relocating graphics files](#) on page 746

§40.3 [Specifying custom settings for individual graphics](#) on page 752

*See also:*

§15.7 [Managing graphics for print RTF](#) on page 234

§17.6 [Managing graphics for WinHelp](#) on page 292

§32 [Including graphics in HTML](#) on page 611

§44.7 [Placing graphics files for distribution](#) on page 796

## 40.1 Choosing an appropriate graphics format

Some graphics formats work better than others in each kind of output. For best appearance, you might have to prepare an alternate set of graphics files, and set some configuration-file options.

*In this section:*

§40.1.1 [Graphics formats for Word documents](#) on page 745

§40.1.2 [Graphics formats for WinHelp](#) on page 745

§40.1.3 [WMF format limitations](#) on page 746

§40.1.4 [Graphics formats for HTML](#) on page 746

### 40.1.1 Graphics formats for Word documents

**WMF or BMP** Use WMF or BMP for bitmap graphics for Word. If you choose WMF, see §40.1.3 [WMF format limitations](#) on page 746.

**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.

### 40.1.2 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 §40.1.3 [WMF format limitations](#) on page 746.

For very large graphics, BMP can be a better choice.

**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.

### 40.1.3 WMF format limitations

WMF graphics are like scripts for the Windows GDI. A WMF graphic can include vector elements, text, and bitmaps (with up to 24-bit resolution).

The WMF graphic format has limitations:

[Bezier curves become polylines](#)

[Dashed line width is ignored](#)

*Bezier curves  
become polylines*

WMF does not support Bezier curves. WMF does support ellipses and elliptic arcs, with radii or chords available for the arcs. Bezier curves are represented by polyline segments.

*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.

### 40.1.4 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:*

§32 [Including graphics in HTML](#) on page 611

## 40.2 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 **DITA2Go** to look for them in a directory different from the location referenced in DITA.

Changing referenced file names and locations for **DITA2Go** requires different settings, depending on whether you are converting to RTF (Word or WinHelp) or to HTML.

This section discusses the following topics:

§40.2.1 [Changing graphics files for HTML output](#) on page 746

§40.2.2 [Changing graphics files for RTF output](#) on page 748

### 40.2.1 Changing graphics files for HTML output

*In this section:*

§40.2.1.1 [Specifying graphics location for HTML](#) on page 747

§40.2.1.2 [Substituting graphics files for HTML](#) on page 747

§40.2.1.3 [Overriding path specifications for referenced graphics](#) on page 748

*See also:*

§32 [Including graphics in HTML](#) on page 611

### 40.2.1.1 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 DITA 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 §32.1 [Locating graphics files for HTML](#) on page 611.

*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 **DITA2Go** copy the graphics after conversion; see §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796.
- Copy the graphics yourself, outside of the conversion process.

See also:

§44.7.4 [Synchronizing graphics settings for HTML output](#) on page 799

§32.1 [Locating graphics files for HTML](#) on page 611

§18.3.9 [Locating graphics files for HTML Help](#) on page 318

§19.3.8 [Getting OmniHelp supporting files in the right place](#) on page 360

§20.3.6.3 [Locating graphics files for JavaHelp and Oracle Help](#) on page 391

### 40.2.1.2 Substituting graphics files for HTML

You can tell **DITA2Go** to use a specific named graphic in place of the original 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 DITA 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

```

### 40.2.1.3 Overriding path specifications for referenced graphics

To override path settings in [GraphFiles] and in configuration markers (see §42.2.9.4 [Overriding graphic properties for HTML](#) on page 774):

```

[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, **DITA2Go** uses the path to graphics specified by GraphPath (see §32.1 [Locating graphics files for HTML](#) on page 611) instead of any path (or lack of a path) specified in [GraphFiles] (see §40.2.1.2 [Substituting graphics files for HTML](#) on page 747) or in a **\*Config** PI marker that has content:

```
[GraphFiles]=filename
```

Also, **DITA2Go** replaces with underscores any spaces in file names of referenced graphics.

You can have **DITA2Go** eliminate spaces from the names of those original files, to make the names valid in all environments:

```

[Graphics]
; FixGraphSpaces = Yes (default, replace space with underscore) or No
FixGraphSpaces = Yes

```

## 40.2.2 Changing graphics files for RTF output

For print RTF or WinHelp output, you can direct **DITA2Go** to use graphics files different from those referenced in DITA, or exclude graphics altogether.

*In this section:*

- §40.2.2.1 [Substituting graphics files for RTF](#) on page 748
- §40.2.2.2 [Using already converted graphics for RTF](#) on page 751
- §40.2.2.3 [Excluding graphics from RTF output](#) on page 751

### 40.2.2.1 Substituting graphics files for RTF

If any graphics referenced in your DITA document are not BMP or WMF, unless you map those graphics to replacements, **DITA2Go** puts them in an INCLUDEPICTURE field for Word output, and omits them from WinHelp output.

*In this section:*

- §40.2.2.1.1 [Substituting files with different extensions](#) on page 748
- §40.2.2.1.2 [Substituting files with different names or locations](#) on page 749
- §40.2.2.1.3 [Understanding replacement examples](#) on page 750

#### 40.2.2.1.1 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 §40.2.2.1.2 [Substituting files with different names or locations](#) on page 749.

#### 40.2.2.1.2 Substituting files with different names or locations

You can tell **DITA2Go** 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 40-1](#) shows where **DITA2Go** expects to find replacement files for various combinations of FileNames and FilePaths values and [GraphFiles] settings.

**Table 40-1 RTF replacement graphics file mappings and locations**

FileNames	Valid [GraphFiles] mappings		Replacement directory when FilePaths =	
	Original graphics file(s)	= Replacement file(s)	Retain	None
<b>Retain</b>	<i>Ignored</i>		Original	Output
<b>Map</b>	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	

*FileNames=Map*

When FileNames=Map, **DITA2Go** uses the settings in [GraphFiles] to find replacements.



<i>FileNames=Retain</i>	When <code>FileNames=Retain</code> , <b>DITA2Go</b> ignores the settings in <code>[GraphFiles]</code> , and looks for replacements in one of two places (determined by the <code>FilePaths</code> setting): the same directory as the original graphics, or the project directory.
<i>FilePaths=None</i>	<p>When <code>FilePaths=None</code>, <b>DITA2Go</b> ignores the path component of the file references in DITA. Unless you specify <code>FileNames=Map</code> and a different path in <code>[GraphFiles]</code>, <b>DITA2Go</b> looks for replacements <i>only in the project directory</i>.</p> <p>You can use <code>FilePaths=None</code> 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 <code>FilePaths=Retain</code> , unless you specify <code>FileNames=Map</code> and different paths for both original and replacement files in <code>[GraphFiles]</code> , <b>DITA2Go</b> 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 <code>FilePaths=None</code> , and put the replacements in the project directory. This is because specifying original file paths in <code>[GraphFiles]</code> is problematic; success depends on exactly matching the paths in DITA, whether they are absolute or relative.

#### 40.2.2.1.3 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 DITA 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.

#### 40.2.2.2 Using already converted graphics for RTF

To instruct **DITA2Go** 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, d2rtf.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)
```

#### 40.2.2.3 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 **DITA2Go** 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
```

## 40.3 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 for an individual graphic, with a PI marker.

You can use PI markers to insert configuration overrides in your DITA document. For RTF output, use a **Config** or **RTFConfig** PI marker for this purpose; for HTML output, a **Config** or **HTMConfig** PI marker. See §42.2.2 [Overriding settings with configuration PI markers](#) on page 767. The following tables show which settings can be overridden:

Table 42-2 [Fixed-key configuration sections subject to overrides](#) on page 770

Table 42-6 [HTML graphic sections subject to overrides](#) on page 775

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 §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769).

*See also:* To override a setting for a given graphic, place the marker just before the `<image>` tag. For marker text, supply the setting you want to change.

§38 [Working with processing instructions](#) on page 717

§42.2 [Overriding settings with PI markers or macros](#) on page 766

# 41 Working with content models

---

**DITA2Go** 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:

- §41.1 [Understanding DITA2Go content models](#) on page 753
- §41.2 [Modifying or replacing a content model](#) on page 753
- §41.3 [Preparing a content model for use with DITA2Go](#) on page 755
- §41.4 [Understanding content-model configurations](#) on page 756
- §41.5 [Understanding how DITA2Go uses content models](#) on page 759
- §41.6 [Inspecting and correcting element types](#) on page 760
- §41.7 [Specializing or modifying DITA topic types](#) on page 761
- §41.8 [Extracting content-model debug information](#) on page 764

*See also:*

- §24 [Converting to DITA XML](#) on page 455
- §26 [Converting to DocBook XML](#) on page 499
- §C [Content model configuration](#) on page 833

## 41.1 Understanding DITA2Go content models

A **DITA2Go** content model is a configuration-style representation of a DTD. A content-model configuration summarizes DTD information in a form **DITA2Go** can use to produce XML output that conforms to the DTD. **DITA2Go** 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 **DITA2Go** conversion project to use these content models.

The **DITA2Go** 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 §41.6 [Inspecting and correcting element types](#) on page 760. 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 **dtd2ini** to abstract content models from this and other DTDs; see §41.2.2 [Generating a content model from a DTD](#) on page 754.

## 41.2 Modifying or replacing a content model

To modify a **DITA2Go** 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:*

§41.2.1 [Obtaining a copy of a built-in content-model](#) on page 754

§41.2.2 [Generating a content model from a DTD](#) on page 754

## 41.2.1 Obtaining a copy of a built-in content-model

If you need to modify one of the **DITA2Go** built-in content models to correct an element type assignment (see §41.6 [Inspecting and correcting element types](#) on page 760), 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	dita11contentmods.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 41-1](#) lists the configuration files in each archive. Extract from the relevant archive the content-model configuration file you wish to modify.

**Table 41-1 Configuration files for DITA2Go built-in content models**

<b>Content model archive</b>	<b>Content model configurations</b>	<b>DTD-to-model configurations</b>
dita10contentmods.zip	ditaconcept10.ini	dtd2concept10.ini
	ditamap10.ini	dtd2map10.ini
	ditareference10.ini	dtd2reference10.ini
	ditatask10.ini	dtd2task10.ini
	ditatopic10.ini	dtd2topic10.ini
dita11contentmods.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

## 41.2.2 Generating a content model from a DTD

Utility program **dtd2ini** can produce, from a valid DTD, a content-model configuration file to use with **DITA2Go** 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 **DITA2Go** distribution. However, you can download **dtd2iniNNwin.zip** from the Omni Systems Web site:

<http://www.dita2go.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<sub>type</sub>.ini</code> , where <i>topic<sub>type</sub></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.

## 41.3 Preparing a content model for use with DITA2Go

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 **DITA2Go**:

1. Inspect and (if necessary) change element type assignments; see §41.6 [Inspecting and correcting element types](#) on page 760.
2. **For DITA only**, if you are adding or replacing a content model, provide information needed by **DITA2Go** that is not available in the DTD:
  - Most settings in section `[Topic]` except for `TopicRoot`; see §41.4.1 [Content model \[Topic\] settings](#) on page 757.

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

<u>Output type</u>	<u>Built-in content model to be replaced by current model</u>
DITA 1.1	topic, concept, task, reference, map, glossary, or bookmap
DocBook	book or article

If you assign any other value to `ModelName`, **DITA2Go** 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
```

## 41.4 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 §41.4.1 <a href="#">Content model [Topic] settings</a> on page 757.
<code>[ElementSets]</code>	Groups elements into sets for assignment in sections <code>[TopicParents]</code> and <code>[TopicFirst]</code> . See §41.4.2 <a href="#">Content model [ElementSets] settings</a> on page 758
<code>[TopicParents]</code>	Lists the valid parent element(s) of each element. See §41.4.3 <a href="#">Content model [TopicParents] settings</a> on page 758.
<code>[TopicFirst]</code>	Lists parent elements for which a given element must be the first child. See §41.4.4 <a href="#">Content model [TopicFirst] settings</a> on page 758.
<code>[TopicLevels]</code>	Specifies required levels for certain elements. See §41.4.5 <a href="#">Content model [TopicLevels] settings</a> on page 759.



[ElementTypes] Classifies each element as to whether it is block or inline, whether it allows text, and whether it is preformatted. See §41.6 [Inspecting and correcting element types](#) on page 760.

*In this section:*

§41.4.1 [Content model \[Topic\] settings](#) on page 757

§41.4.2 [Content model \[ElementSets\] settings](#) on page 758

§41.4.3 [Content model \[TopicParents\] settings](#) on page 758

§41.4.4 [Content model \[TopicFirst\] settings](#) on page 758

§41.4.5 [Content model \[TopicLevels\] settings](#) on page 759

## 41.4.1 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"
```

**DITA2Go** 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. **DITA2Go** 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 §41.7 [Specializing or modifying DITA topic types](#) on page 761.

### 41.4.2 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.

### 41.4.3 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 §41.4.2 [Content model \[ElementSets\] settings](#) on page 758.

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.

### 41.4.4 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 §41.4.2 [Content model \[ElementSets\] settings](#) on page 758.

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 DITA document **DITA2Go** 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.

### 41.4.5 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:

§24.5.12 [Specifying DITA element levels](#) on page 479

§26.5.11 [Specifying DocBook element levels](#) on page 518

## 41.5 Understanding how DITA2Go 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 §41.1 [Understanding DITA2Go content models](#) on page 753. 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

```

**DITA2Go** uses a complex algorithm to determine which element to interpolate in places in your document where a parent element is required. When **DITA2Go** processes your document and encounters text that you have mapped to a `<data>` element (for example), **DITA2Go** searches the above element sets, in sequence, for the current parent element. If the parent is not found, **DITA2Go** performs a graph analysis, breadth-first, of possible parent series that could fit under the current parent. In each case, **DITA2Go** 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, **DITA2Go** also considers closing existing parents to find a better solution to the graph problem.)

Suppose you want to tell **DITA2Go** *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.)

## 41.6 Inspecting and correcting element types

Utility program **dtd2ini** 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 **dtd2ini**, include any changed [ElementTypes] settings as overrides in configuration file `dtd2ini.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 PI 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 §24.4.3.1 [Assigning DITA elements to paragraph formats](#) on page 461. That way you will not be forced to use CodeBefore and CodeAfter settings to insert the tags for such an element.

## 41.7 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 §41.2.2 [Generating a content model from a DTD](#) on page 754) on a local document type shell, and reference the result in your project configuration chain.

*In this section:*

- §41.7.1 [Creating a content model for a specialized topic type](#) on page 761
- §41.7.2 [Overriding settings in a DITA content model](#) on page 762
- §41.7.4 [Overriding declarations in a DITA map content model](#) on page 763
- §41.7.5 [Listing DITA topic type configuration files](#) on page 763
- §41.7.6 [Locating DITA topic type configuration files](#) on page 764

### 41.7.1 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 §41.2.2 [Generating a content model from a DTD](#) on page 754.
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 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 §24.5.12 [Specifying DITA element levels](#) on page 479.

3. In your project configuration file, list the name of your new topic type:

```
[DITAContentModels]
DITAnewtype = any text here (ignored)
```

See §41.7.5 [Listing DITA topic type configuration files](#) on page 763.

4. Place `DITAnewtype.ini` where **DITA2Go** can find it; see §41.7.6 [Locating DITA topic type configuration files](#) on page 764.

## 41.7.2 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 §41.7.4 [Overriding declarations in a DITA map content model](#) on page 763.

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 bookmap), or any specialized type for which a content-model configuration file named `DITAtopictype.ini` is available (see §41.7 [Specializing or modifying DITA topic types](#) on page 761). *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 §41.7.5 [Listing DITA topic type configuration files](#) on page 763.

5. Place `DITAtopictype.ini` where **DITA2Go** can find it; see §41.7.6 [Locating DITA topic type configuration files](#) on page 764.

For example, to change the PUBLIC declaration for glossary topics (to conform to XMetaL 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)
```

### 41.7.3 Eliminating elements from a DITA content model

If you want to be able to tell **DITA2Go** *not* to use certain elements, 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.

### 41.7.4 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 §41.7.2 [Overriding settings in a DITA content model](#) on page 762. However, for the maps **DITA2Go** 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 §41.7.5 [Listing DITA topic type configuration files](#) on page 763.

### 41.7.5 Listing DITA topic type configuration files

When you provide a `DITAtopicitype.ini` configuration file, you must list the name of the topic type in your project configuration file, so **DITA2Go** 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]
DITAtopicitype = 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; **DITA2Go** reads only the topic type name to the left of the equals sign.

Provide a `DITAtopicitype.ini` configuration file named for each topic type you list; see §41.2.2 [Generating a content model from a DTD](#) on page 754. or §41.7.2 [Overriding settings in a DITA content model](#) on page 762. **DITA2Go** loads each listed `DITAtopicitype.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 §24.8.2.2 [Specifying a default DITA topic type](#) on page 486), or in a **DITATopic** PI marker, in which case the corresponding



DITA*topic*type.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.

### 41.7.6 Locating DITA topic type configuration files

By default, **DITA2Go** 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.

## 41.8 Extracting content-model debug information

You can have **DITA2Go** 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 DITA 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, **DITA2Go** places the dump file in the project directory.

# 42 Overriding configuration settings

---

You can provide different configuration settings for individual ditamap files, and you can also override configuration settings for one or more paragraphs, character spans, tables, graphics, or cross references assigned to DITA elements. Topics include:

§42.1 [Using a different configuration for selected files](#) on page 765

§42.2 [Overriding settings with PI markers or macros](#) on page 766

§42.3 [Overriding configuration settings with text](#) on page 776

*See also:*

§31.6.2 [Changing CSS files in the middle of a document](#) on page 598

## 42.1 Using a different configuration for selected files

If you need different configuration settings for one or more ditamap files, you can create individual, file-specific configuration files.

*In this section:*

§42.1.1 [Providing configuration files for individual ditamaps](#) on page 765

§42.1.2 [Understanding precedence of configuration settings](#) on page 765

### 42.1.1 Providing configuration files for individual ditamaps

To provide individual configuration files:

- Name each configuration file the same as the ditamap file name, with extension `.ini`.
- Place individual configuration files in the same directory as the main configuration file for the project.
- Include in these ditamap-specific configuration files *only* those settings that are different from settings in the main project configuration file.

When you run **DITA2Go** from the top-level map 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 ditamap file.

### 42.1.2 Understanding precedence of configuration settings

At run time **DITA2Go** builds a configuration for the starting map file in your project, beginning with the most specific settings: those in any map-specific configuration file, if there is one. Next come settings in the project configuration file.

*Chain of  
configuration  
templates*

Next, if the map-specific configuration file includes a value for `[Templates]Configs` (see §39.2 [Referencing configuration files and templates](#) on page 731), settings in the referenced configuration template (and any additional templates chained to it) are applied. If the map-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 42-1](#) shows the precedence of settings in configuration files and templates.

**Table 42-1 Precedence of settings in configuration files and templates**

Precedence	Configuration file	Description
Highest	<i>ditamap.ini</i>	Configuration file (if any) for a single DITA map file
	<i>_d2*.ini</i>	Project configuration file
	<i>chaptemplate.ini</i> or	Template referenced by <i>ditamap.ini</i> , if any
	<i>doctemplate.ini</i>	Template referenced by <i>_d2*.ini</i> via [Templates]Document if no such template is referenced by <i>ditamap.ini</i> (or no <i>ditamap.ini</i> is present)
	<i>projtemplate.ini</i>	Template referenced by <i>_d2*.ini</i> if no template is referenced by <i>ditamap.ini</i> (or no <i>ditamap.ini</i> is present)
	<i>commontemplate1.ini</i>	Template referenced by <i>chaptemplate.ini</i> or by <i>projtemplate..ini</i> , whichever is used
Lowest	<i>commontemplateN.ini</i>	Template referenced by <i>commontemplateN-1.ini</i>
	Default value	Whatever the <b>DITA2Go</b> default value is for the setting in question

A chain of configuration templates, if any, is applied to the source either from *ditamap.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 §39.5.3 [Chaining configuration templates](#) on page 743.

## 42.2 Overriding settings with PI markers or macros

To change the value of a configuration setting partway through a DITA file, you assign a new value to a configuration variable. You can insert a PI marker that contains the assignment or (for some settings) define a **DITA2Go** macro that includes the assignment. Both methods allow you to shift configuration values back and forth within the same DITA file.

*In this section:*

- §42.2.1 [Determining the extent of a configuration override](#) on page 766
- §42.2.2 [Overriding settings with configuration PI markers](#) on page 767
- §42.2.3 [Overriding settings with macros](#) on page 767
- §42.2.4 [Assigning values to configuration variables](#) on page 768
- §42.2.5 [Adding a new configuration setting on the fly](#) on page 768
- §42.2.6 [Assigning a macro or variable to a configuration variable](#) on page 768
- §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769
- §42.2.8 [Overriding fixed-key configuration settings](#) on page 770
- §42.2.9 [Overriding variable-key configuration settings](#) on page 771
- §42.2.10 [Assigning HTML table and graphic groups with overrides](#) on page 775

### 42.2.1 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 §42.2.4 [Assigning values to configuration variables](#) on page 768. **DITA2Go** does not store either persistent or temporary overrides in your configuration file. The configuration file always retains the original values of the settings.

*Persistent overrides*

A *persistent override* stays in effect until changed by another override of the same setting, or until the end of the DITA file in which the override occurs, whichever comes first. To apply a persistent override, insert a PI marker in your document just before the place where you want the override to take effect; and (optionally) later, another PI marker to reverse the effect. For certain fixed-key settings, you can include a configuration override in a regular **DITA2Go** macro instead of in a PI marker; see §42.2.3 [Overriding settings with macros](#) on page 767.

*Temporary overrides*

A *temporary override* affects only one instance of the item (text, table, or graphic) to which the setting applies. To apply a temporary override, you insert a PI marker just before the item to which the override should apply. Temporary overrides can be applied only to variable-key settings; see §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769.

## 42.2.2 Overriding settings with configuration PI markers

To change a configuration setting mid-document with a configuration PI marker, you can use one of the following PI marker types:

<b>Config</b>	applies either to HTML or to RTF, wherever the setting is applicable
<b>HTMConfig</b>	applies only to HTML output; ignored for RTF output
<b>RTFConfig</b>	applies only to RTF output; ignored for HTML output.

To change the value of a configuration setting partway through your document, insert a configuration PI 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 §42.2.4 [Assigning values to configuration variables](#) on page 768.

## 42.2.3 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 PI 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:

<b>HTML Code</b> PI marker	HTML output only; ignored for RTF output
<b>Code</b> PI marker	HTML or RTF output, wherever the setting is applicable
<b>DITA2Go</b> 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 PI marker, insert a PI 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 §42.2.4 [Assigning values to configuration variables](#) on page 768.

For persistent overrides only, you can include the configuration-variable assignment in a configuration macro that applies the directive based on some condition; see §42.2.1 [Determining the extent of a configuration override](#) on page 766.

## 42.2.4 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>
<b>*Config</b> PI marker	[ <i>Section</i> ]Key=Value	[ <i>Section</i> ]=Value
<b>DITA2Go</b> 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** PI marker, the prefix is optional.
- Include *Key* only for a persistent override; see §42.2.1 [Determining the extent of a configuration override](#) on page 766.
- *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, **DITA2Go** 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** PI marker, quotes around text values are optional; if present, **DITA2Go** removes them. Therefore, if the value to be assigned actually contains quotes at both ends, you must double them for assignment in a **\*Config** PI 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 §42.2.6 [Assigning a macro or variable to a configuration variable](#) on page 768.

## 42.2.5 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 42-2](#) through [Table 42-6](#).
- *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).

## 42.2.6 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** PI 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 **DITA2Go** processes each angle bracket as the start or end of a macro, instead of assigning the entire value as a string. That is, **DITA2Go** 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>>

**DITA2Go** would assign only <hr to the configuration variable, because the > after <hr would be taken as the end of the macro; and then **DITA2Go** 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.

## 42.2.7 Understanding fixed-key vs. variable-key settings

The settings in some **DITA2Go** 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 §42.2.1 [Determining the extent of a configuration override](#) on page 766. [Table 42-2](#) on page 770 lists the fixed-key configuration sections that include settings subject to override.

*Variable-key  
configuration  
sections*

Configuration sections such as [HelpStyles] 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 42-3 [Text configuration sections subject to overrides](#) on page 772

Table 42-4 [Cross-reference sections subject to overrides](#) on page 773

Table 42-5 [HTML table sections subject to overrides](#) on page 774

Table 42-6 [HTML graphic sections subject to overrides](#) on page 775

## 42.2.8 Overriding fixed-key configuration settings

An override to a fixed-key configuration setting stays in effect until the end of the current DITA file, or until changed again by another configuration PI marker or configuration-variable assignment to the same setting. You can override some (but not all) settings in the configuration sections listed in [Table 42-2](#) on page 770. 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 DITA 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 **DITA2Go** processes the file content. Other settings such as [WordOptions]SideHeads affect margins, and must apply to an entire file.

If you are producing HTML output, the only way to specify attributes for <body> is with a configuration override. For example, placing a PI marker at the beginning of the second topic:

```
<?dthtm config="[Attributes]body= onload='prettyPrint()' " ?>
```

Then at the beginning of the fourth topic:

```
<?dthtm config="[Attributes]body=" ?>
```

The effect would appear in HTML output for the second and third topics, but not the fourth. The PI marker affects the output file for the topic in which it is included (right after the root), and continues until set otherwise.

You can also override a fixed-key setting with a configuration-variable assignment in a regular **DITA2Go** macro instead of in a PI marker. See §37.9.3 [Surrounding or replacing text with code or macros](#) on page 711.

**Table 42-2 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
[Graphics]	Yes	Yes	Yes
[HelpBrowse]			Yes
[HelpContents]			Yes
[HelpOptions]			Yes
[HTMLOptions]	Yes		

\* Some settings cannot be overridden in these sections; you might have to experiment.



**Table 42-2 Fixed-key configuration sections subject to overrides (continued)**

Fixed-key configuration section *	HTML/XML	Word	WinHelp
[Inserts]	Yes	Yes	Yes
[JavaHelpOptions]	Yes		
[Macros]	Yes		
[MSHtmlHelpOptions]	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.

## 42.2.9 Overriding variable-key configuration settings

*In this section:*

§42.2.9.1 [Overriding paragraph and character format properties](#) on page 771

§42.2.9.2 [Overriding cross-reference properties](#) on page 773

§42.2.9.3 [Overriding table properties for HTML](#) on page 773

§42.2.9.4 [Overriding graphic properties for HTML](#) on page 774

### 42.2.9.1 Overriding paragraph and character format properties

You can override character and paragraph format settings in the configuration sections listed in [Table 42-3](#). For example, to specify new properties for a single paragraph in HTML, you could insert in the paragraph an **HTMConfig** PI marker with content different from the default:

```
<?dthtm HTMConfig="[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 PI marker with respect to paragraph and character formats is critical:

- A temporary-override PI marker placed in a block element affects the entire element, including any contained inline elements, and therefore any formats assigned to the current instance of the block or inline elements.
- A temporary-override PI marker placed in an inline element affects only the character format assigned to that instance of the element.

*Persistent  
overrides*

A persistent override affects the next instance of an element to which the paragraph or character format named by the *Key* in `[Section]Key=Value` is assigned, or the current instance if the PI marker is placed in an element to which a matching paragraph or character format is assigned; plus all subsequent instances in the same DITA file, unless changed again by a later override.

*PI 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 PI marker in the replaced text is

ignored. This means you cannot use a temporary override in a configuration PI marker for the replacement; instead you must use a persistent override that names the format to be replaced, and insert the configuration PI 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 element to which the paragraph or character is assigned. Persistent overrides should be placed before the affected block or inline element.

**Table 42-3 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
[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
[HelpReplacements]			Yes
[HelpStyles]			Yes
[HelpSuffixStyles]			Yes
[HelpTitleSufStyles]			Yes
[HelpTopicBuildStyles]			Yes
[HelpWindowStyles]			Yes
[HTMLCharStyles]	Yes		
[HTMLParaStyles]	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		

**Table 42-3 Text configuration sections subject to overrides (continued)**

Text configuration section	HTML/XML	Word	WinHelp
[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		
[StyleTitlePrefix]	Yes		
[StyleTitleSuffix]	Yes		
[StyleTrailPrefix]	Yes		
[StyleTrailSuffix]	Yes		
[Targets]	Yes		
[TrailLevels]	Yes		
[WordReplacements]		Yes	
[WordStyles]		Yes	

### 42.2.9.2 Overriding cross-reference properties

You can use configuration PI markers and configuration-variable assignments to override settings in [XrefStyles] and in the HTML [XrefStyleLinkSrc] section; see [Table 42-4](#).

A temporary override to a cross-reference format affects the next cross reference after the configuration PI 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 DITA file, unless changed again by a later override.

**Table 42-4 Cross-reference sections subject to overrides**

Cross-reference section	HTML/XML	Word	WinHelp
[XrefStyleLinkSrc]	Yes		
[XrefStyles]	Yes	Yes	Yes

### 42.2.9.3 Overriding table properties for HTML

You can use configuration PI markers and configuration-variable assignments to override settings in the HTML [Table\*] sections listed in [Table 42-5](#).

A temporary override to a table affects the entire table within which a configuration PI marker is placed; or the next table, if the PI 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 PI

marker is placed in a matching table. For table formats and groups, the override also affects all subsequent matching instances in the same DITA file, unless changed again by a later override.

**Table 42-5 HTML table sections subject to overrides**

---

**Table configuration section**

---

```
[TableAccess]
[TableAfterMacros]
[TableAttributes]
[TableBeforeMacros]
[TableBodyAttributes]
[TableCellAttributes]
[TableCellEndMacros]
[TableCellStartMacros]
[TableEndMacros]
[TableFooterAttributes]
[TableGroup]
[TableHeaderAttributes]
[TableReplaceMacros]
[TableRowAttributes]
[TableRowEndMacros]
[TableRowStartMacros]
[TableSizing]
[TableStartMacros]
```

---

*See also:*

[§42.2.10 Assigning HTML table and graphic groups with overrides](#) on page 775

#### 42.2.9.4 Overriding graphic properties for HTML

You can use configuration PI markers and configuration-variable assignments to override any variable-key settings and some fixed-key settings in the HTML [Graph\*] sections listed in [Table 42-6](#).

A temporary override to a graphic affects the next graphic after the point in your document where you insert the PI marker.

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 DITA 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** PI marker, not a macro, to specify the graphic group.

*Overriding the  
overrides*

To override path settings both in [GraphFiles] and in configuration PI 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`, **DITA2Go** uses the path to graphics specified by `GraphPath` (see §40.2.1.1 [Specifying graphics location for HTML](#) on page 747) instead of any path (or lack of a path) specified in `[GraphFiles]` (see §40.2.1.2 [Substituting graphics files for HTML](#) on page 747) or in a **\*Config** PI marker with content:

```
[GraphFiles]=filename
```

Also, **DITA2Go** replaces with underscores any spaces in file names of referenced graphics.

**Table 42-6 HTML graphic sections subject to overrides**

---

**Graphic configuration section**

---

```
[GraphALT]
[GraphAttr]
[GraphEndMacros]
[GraphFiles]
[GraphHigh]
[GraphReplaceMacros]
[GraphScale]
[GraphStartMacros]
[GraphWide]
```

---

*See also:*

§32.6 [Specifying HTML image attributes](#) on page 619

§42.2.10 [Assigning HTML table and graphic groups with overrides](#) on page 775

## 42.2.10 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 PI marker assigns the table or graphic to a named group. You can use only a temporary override applied with a **\*Config** PI marker, not a macro, to specify the group name. Any key included in the PI marker is ignored.

*Table groups* If you put a `[TableGroup]` configuration PI 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.

**Note:** Each table can belong to only one table group.

*See also:*

§33.2.1 [Creating table groups](#) on page 626

§42.2.9.3 [Overriding table properties for HTML](#) on page 773

*Graphic groups* If you put a `[GraphGroup]` configuration PI 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.

**Note:** Each graphic can belong to only one graphic group.

*See also:*

§32.4.1.2 [Using PI markers to assign properties to graphics](#) on page 614

§42.2.9.4 [Overriding graphic properties for HTML](#) on page 774

## 42.3 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** PI markers in your document, and it works the same way. See §42.2.2 [Overriding settings with configuration PI markers](#) on page 767.

To make a 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**, **DITA2Go** 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 §42.2.7 [Understanding fixed-key vs. variable-key settings](#) on page 769.

You have two choices of syntax for **\*Config** paragraph content; you can intermix them in the same paragraph:

*File syntax:* 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 §3.4 [Understanding the rules for configuration settings](#) on page 62.

*Marker syntax:* Use the same syntax as for **\*Config** PI markers; see §42.2.4 [Assigning values to configuration variables](#) on page 768. 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 PI markers at the same location in your document.

# 43 Automating DITA2Go conversions

---

**DITA2Go** supports several techniques for automating conversion workflow. This section includes the following topics:

§43.1 [Executing operating-system commands](#) on page 777

§43.2 [Converting autonumbers for database systems](#) on page 780

§43.3 [Renaming output files for automated systems](#) on page 781

*See also:*

§44 [Producing deliverable results](#) on page 787

§45 [Converting via DCL](#) on page 809

## 43.1 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. **DITA2Go** can perform these kinds of chores automatically by executing operating-system commands that you specify in the project configuration file.

§43.1.1 [Specifying system commands](#) on page 777

§43.1.3 [Monitoring system command execution](#) on page 778

§43.1.4 [Supplying system commands in a .bat file](#) on page 779

§43.1.5 [Supplying system commands in a macro](#) on page 779

### 43.1.1 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 §44.2 [Activating and logging production of deliverables](#) on page 788):

```
[Automation]
SystemEndCommand = rename .\wrap\ugdita2go.htm _ugdita2go.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:

§43.1.4 [Supplying system commands in a .bat file](#) on page 779

§43.1.5 [Supplying system commands in a macro](#) on page 779.



When you assign a system command (or a macro) to a `System*Command`, **DITA2Go** generates one or more lines of code, each of which is a command to be run at a Windows command prompt. **DITA2Go** 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 §44.13 [Postprocessing separately from converting](#) on page 807.

### 43.1.2 Including macros and variables in system commands

System commands can include the following:

- **DITA2Go** macro expressions
- macro variables you have defined in `[MacroVariables]`.
- the following predefined macro variables (see §37.3.4 [Using predefined macro variables](#) on page 691):

<code>&lt;\$\$_basename&gt;</code>	Base file name (without path or extension) of the current DITA source file
<code>&lt;\$\$_currpath&gt;</code>	Path (without trailing slash) to the current directory where the configuration file resides
<code>&lt;\$\$_macroparam&gt;</code>	Value of a parameter passed to the enclosing macro.
<code>&lt;\$\$_prjpath&gt;</code>	Path (without trailing slash) to the directory where the map 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 §37.7 [Passing a parameter to a macro](#) on page 709.

*Include macro  
expressions*

You can use macro expressions in system commands: math and string manipulations, conditional expressions, loops, formatted output, and so forth; see §37.6 [Using expressions in macros](#) on page 700.

### 43.1.3 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, **DITA2Go** recycles the .bat file.

### 43.1.4 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 **DITA2Go** variables in .bat files.

### 43.1.5 Supplying system commands in a macro

You can put system commands in a **DITA2Go** 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 §37.1 [Defining and invoking macros](#) on page 679.

For example, suppose your workflow requires backing up your DITA 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 <$$_currpath>\\*.dita x:\\backup
copy <$$_currpath>\\*.dita "y:\\my other\\backup"
```

Notice the doubled backslashes (required in **DITA2Go** macros, where backslash is used as an escape character), and the quotes around the path that includes a space. See §37.1.1 [Defining macros](#) on page 679.

*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 <$$_currpath>\\*.dita x:\\backup
; copy <$$_currpath>\\*.dita "y:\\my other\\backup"
```

## 43.2 Converting autonumbers for database systems

Suppose you use autonumbers for headings of the style you see in the **DITA2Go User's Guide**; and suppose you use an automated system to populate a database with the number and text of each heading, from **DITA2Go**-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 `<a name=nnn>`.

For example, suppose you have three heading format levels. 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, **DITA2Go** sets aside the generated HTML code in macro variable `$$Stored`. **DITA2Go** parses the stored code as follows, to extract the autonumber:

1. Skips everything in the stored code up through the `<b>` tag.
2. Puts in `$$Anum` everything between the `<b>` tag and the next space; this includes the whole autonumber.
3. Stores in `$$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]`, **DITA2Go** puts out the start of the `<a>` tag, and then whichever of the macro variables is needed. Finally, **DITA2Go** adds the original stored heading itself to the output, and closes the `<a>` tag.

See §37 [Working with macros](#) on page 679.

## 43.3 Renaming output files for automated systems

For names of **DITA2Go**-generated files, **DITA2Go** 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 26.

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 **DITA2Go** breaks any links to that file; however, see §28.6.2 [Enabling links to renamed or relocated files](#) on page 553.

*In this section:*

§43.3.1 [Renaming individual output files](#) on page 781

§43.3.2 [Using PI markers to name output files](#) on page 782

§43.3.3 [Using paragraph formats to name output files](#) on page 782

### 43.3.1 Renaming individual output files

*Do not try to rename **DITA2Go**-generated files outside of **DITA2Go**.*

To substitute a different name for a particular output file, map the original name to the new name (without extension):

```
[HtmlFiles]
; original html filename = desired html filename
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 DITA name. **DITA2Go** writes this file even if it is essentially empty.

See also:

§27.5.1 [Understanding how split and extract files are named](#) on page 530.

### 43.3.2 Using PI markers to name output files

To specify a file name via PI marker, you can use a PI marker type called **FileName** and insert a **FileName** PI marker in the first paragraph of each part to be split or extracted. Make the content of each **FileName** PI marker the name you want for the resulting file, without path or extension.

*Duplicated file  
names are hard  
to locate*

Using **FileName** PI 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 PI markers. Expect many problems of this type if you use **FileName** PI markers to override **DITA2Go**-generated file names.

See also:

§38 [Working with processing instructions](#) on page 717

§27 [Splitting and extracting files](#) on page 523

### 43.3.3 Using paragraph formats to name output files

According to **DITA2Go** developers, naming output files using paragraph content is a *Very Bad Idea*. You are almost certain to have name conflicts that result in **DITA2Go** 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:

§43.3.3.1 [Constructing file names based on paragraph content](#) on page 783

§43.3.3.2 [Basing output file names on existing paragraph formats](#) on page 784

§43.3.3.3 [Creating special paragraph formats to name output files](#) on page 784

§43.3.3.4 [Specifying a file-name prefix or suffix](#) on page 784

§43.3.3.5 [Constructing file names from multiple paragraph formats](#) on page 785

§43.3.3.6 [Preventing duplicate file names based on paragraph formats](#) on page 785

### 43.3.3.1 Constructing file names based on paragraph content

You can specify names for HTML or XML output files by designating a 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, **DITA2Go** uses the internally assigned object ID of the paragraph for the file name instead (see §27.5.1 [Understanding how split and extract files are named](#) on page 530), along with any prefix or suffix you specify for file names (see §43.3.3.4 [Specifying a file-name prefix or suffix](#) on page 784).

*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 **DITA2Go** uses the internally assigned object ID 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.

### 43.3.3.2 Basing output file names on existing paragraph formats

If you have assigned a paragraph format to topic titles, 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 §43.3.3.4 [Specifying a file-name prefix or suffix](#) on page 784). This is a simple way to use titles for file names.

However, if you are creating HTML Help (see §18 [Generating Microsoft HTML Help](#) on page 313) 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 a sequence number.

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 §43.3.3.6 [Preventing duplicate file names based on paragraph formats](#) on page 785 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.

### 43.3.3.3 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 element with `@outputclass` named for `ParaFmt` anywhere after a split heading and before the next split point. Although the element can be anywhere in the split file, usually you would put it right after the heading that starts the split. **DITA2Go** 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 §30.2.6 [Eliminating unwanted paragraphs](#) on page 569);.

### 43.3.3.4 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 §37.1 [Defining and invoking macros](#) on page 679) in sections `[StyleFilePrefix]` and `[StyleFileSuffix]`.

### 43.3.3.5 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 §37.3.5.1 [Capturing paragraph content with the TextStore property](#) on page 692), so you can simply specify `$$Heading1` in the prefix value. You can use a macro expression to replace any spaces in *Heading1* content; see §37.6.5 [Specifying substrings in expressions](#) on page 706.

### 43.3.3.6 Preventing duplicate file names based on paragraph formats

To ensure uniqueness of file names without using a prefix or suffix, you can have **DITA2Go** override a file name, by combining the two paragraph-based file-naming methods:

1. Add an element with a special `@outputclass` to enable creation of a file name other than from an existing paragraph, as described in §43.3.3.3 [Creating special paragraph formats to name output files](#) on page 784.
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
```



# 44 Producing deliverable results

---

**DITA2Go** 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:

- §44.1 [Understanding DITA2Go pre- and post-processing](#) on page 787
- §44.2 [Activating and logging production of deliverables](#) on page 788
- §44.3 [Understanding path values for deliverables](#) on page 788
- §44.4 [Clearing out old files before converting](#) on page 789
- §44.5 [Gathering additional files before converting](#) on page 791
- §44.6 [Assembling files for distribution](#) on page 792
- §44.7 [Placing graphics files for distribution](#) on page 796
- §44.8 [Placing CSS or XSL files for assembly](#) on page 800
- §44.9 [Gathering files for an HTML project: an example](#) on page 801
- §44.10 [Gathering and processing Help-system files](#) on page 802
- §44.11 [Archiving deliverables](#) on page 803
- §44.12 [Placing deliverables in a shipping directory](#) on page 806
- §44.13 [Postprocessing separately from converting](#) on page 807

*See also:*

- §43 [Automating DITA2Go conversions](#) on page 777

## 44.1 Understanding DITA2Go pre- and post-processing

When you convert a document, **DITA2Go** 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.

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 **DITA2Go** can handle the compiling or archiving for you.

Before generating output files, **DITA2Go** can do the following:

- 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, **DITA2Go** 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.

## 44.2 Activating and logging production of deliverables

To have **DITA2Go** assemble files and optionally archive deliverables, 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`, **DITA2Go** acts on all options in the `[Automation]` section of the configuration file. To have **DITA2Go** place deliverables in a shipping directory, you must also specify a value for `ArchiveCommand`; see §44.11 [Archiving deliverables](#) on page 803.

When `WrapAndShip=No` (the default), **DITA2Go** ignores most `[Automation]` settings, unless you have **DITA2Go** run a compiler or indexer for a Help system (see §44.10 [Gathering and processing Help-system files](#) on page 802). [Table 44-4](#) on page 803 shows which `[Automation]` settings are activated when **DITA2Go** 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 §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796.

*Log the actions  
taken*

To have **DITA2Go** 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 **DITA2Go** executes to carry out an automation option is recorded in the **DITA2Go** log file, provided logging is enabled; see §4.2 [Logging conversion events](#) on page 74. `LogAuto` takes effect only when `WrapAndShip=Yes`.

## 44.3 Understanding path values for deliverables

When you set up a new conversion project, **DITA2Go** 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 DITA 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:

§44.6.1 [Specifying a wrap directory](#) on page 792

§44.12.1 [Specifying a shipping directory for deliverables](#) on page 806

*See also:*

§3.5 [Specifying file paths in configuration settings](#) on page 64

## 44.4 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 conversion files and old output files in the project directory. You can have **DITA2Go** 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:*

§44.4.1 [Specifying when to delete old files from the project directory](#) on page 789

§44.4.2 [Specifying which files to delete from the project directory](#) on page 790

§44.4.3 [Understanding when not to delete .ref and .htm files](#) on page 791

*See also:*

§44.6.2 [Emptying the wrap directory before copying](#) on page 792

### 44.4.1 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),
; or File (for running a single-file project with no external links).
EmptyOutputDir = Never
```

Values of `EmptyOutputDir` have the following effects:

**Never**     *Default.* **DITA2Go** does not delete any files from the project directory before conversion.

**File**     **DITA2Go** deletes the specified files before conversion.

To determine which files to delete before conversion, see:

§44.4.2 [Specifying which files to delete from the project directory](#) on page 790

§44.4.3 [Understanding when not to delete .ref and .htm files](#) on page 791.

**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 §44.10 [Gathering and processing Help-system files](#) on page 802.

## 44.4.2 Specifying which files to delete from the project directory

To specify which files **DITA2Go** 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 §44.4.1 [Specifying when to delete old files from the project directory](#) on page 789), by default **DITA2Go** 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, **DITA2Go** ignores EmptyOutputDir and logs a warning. To delete .dcl and .dcb files, see §4.1.9 [Reusing or discarding ASCII DCL files](#) on page 73.

*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 §44.4.1 [Specifying when to delete old files from the project directory](#) on page 789), **DITA2Go** deletes the specified files before conversion begins.

*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 §44.10 [Gathering and processing Help-system files](#) on page 802.

See also:

§4.1.9 [Reusing or discarding ASCII DCL files](#) on page 73

§44.4.1 [Specifying when to delete old files from the project directory](#) on page 789

§44.4.3 [Understanding when not to delete .ref and .htm files](#) on page 791

### 44.4.3 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 §27.3 [Splitting files](#) on page 526).

If you are converting to any HTML output type, in the following situation you *must* provide an explicit setting for EmptyOutputFiles that does *not* include \*.ref:

- EmptyOutputDir=File, and you are converting a file that has interfile links.

*External links  
require keeping  
.ref files*

By default, **DITA2Go** deletes \*.ref when EmptyOutputDir is in play (see §44.4.1 [Specifying when to delete old files from the project directory](#) on page 789) and there is no explicit setting for EmptyOutputFiles (see §44.4.2 [Specifying which files to delete from the project directory](#) on page 790). When reference files are removed, you lose the information added from other 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.

*See also:*

§44.4.1 [Specifying when to delete old files from the project directory](#) on page 789

§44.4.2 [Specifying which files to delete from the project directory](#) on page 790

## 44.5 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 **DITA2Go** 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 §44.4 [Clearing out old files before converting](#) on page 789.

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:*

§44.6.6 [Listing extracurricular files to put in the wrap directory](#) on page 795



## 44.6 Assembling files for distribution

**DITA2Go** 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:*

- §44.6.1 [Specifying a wrap directory](#) on page 792
- §44.6.2 [Emptying the wrap directory before copying](#) on page 792
- §44.6.3 [Listing files to copy to the wrap directory](#) on page 793
- §44.6.4 [Understanding when to use other file copy settings](#) on page 794
- §44.6.5 [Understanding which files are copied from where](#) on page 794
- §44.6.6 [Listing extracurricular files to put in the wrap directory](#) on page 795

### 44.6.1 Specifying a wrap directory

To specify a wrap directory where **DITA2Go** 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, **DITA2Go** creates it.

When you first set up a conversion project, by default **DITA2Go** includes the following setting for WrapPath in your new configuration file (see §44.3 [Understanding path values for deliverables](#) on page 788):

```
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, **DITA2Go** 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 §44.10 [Gathering and processing Help-system files](#) on page 802.

### 44.6.2 Emptying the wrap directory before copying

To clear out the wrap directory before **DITA2Go** 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, **DITA2Go** deletes the entire contents of the `WrapPath` directory before copying files.

However, if either of the following is true, **DITA2Go** 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`, **DITA2Go** does not delete files unless both `EmptyWrapPath` and `EmptyGraphPath` are set to `Yes`; see §44.7 [Placing graphics files for distribution](#) on page 796.

When `EmptyWrapPath=No`, **DITA2Go** 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 §44.10 [Gathering and processing Help-system files](#) on page 802.

### 44.6.3 Listing files to copy to the wrap directory

To list files for **DITA2Go** 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 §44.6.1 [Specifying a wrap directory](#) on page 792.

**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 §44.10 [Gathering and processing Help-system files](#) on page 802.

If no setting for `WrapCopyFiles` is present, certain files are copied by default from the project directory to the wrap directory. [Table 44-1](#) lists the files that are copied by default for each output type.

**Table 44-1** 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).

#### 44.6.4 Understanding when to use other file copy settings

Use settings *other than* `WrapCopyFiles` for the following:

- JavaHelp and Oracle Help files; see §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.
- Graphics files and CSS files; see §44.7 [Placing graphics files for distribution](#) on page 796 and §44.8 [Placing CSS or XSL files for assembly](#) on page 800.

#### 44.6.5 Understanding which files are copied from where

If `WrapPath` has a value *other than the project directory* when you run a conversion, by default **DITA2Go** does the following:

- If the directory designated by `WrapPath` already exists, and `EmptyWrapPath=Yes` (the default), **DITA2Go** deletes the prior contents; otherwise **DITA2Go** creates the directory.
- After converting your document, **DITA2Go** copies necessary files from the project directory to the `WrapPath` directory (and to subdirectories, if appropriate). [Table 44-2](#) lists the files that are copied by default.

For example, to have **DITA2Go** 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, **DITA2Go** also copies graphics files and CSS files from the project directory, unless you specify otherwise; see §44.7 [Placing graphics files for distribution](#) on page 796 and §44.8 [Placing CSS or XSL files for assembly](#) on page 800.

**Table 44-2 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 §20.3.6 [Creating a directory structure for JavaHelp / Oracle Help](#) on page 389.

\*\* For OmniHelp, additional files are copied from a viewer directory; see §19.13 [Assembling OmniHelp files for viewing](#) on page 380.

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 §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796.

For all HTML output types, files specified by CssCopyFiles (default \*.css and \*.xsl) are copied from the project directory to the CssPath directory; see §44.8 [Placing CSS or XSL files for assembly](#) on page 800.

## 44.6.6 Listing extracurricular files to put in the wrap directory

If your distribution should include other files in addition to those produced by **DITA2Go**, 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.

## 44.7 Placing graphics files for distribution

If graphics files referenced by your document are not already in the project directory, you can have **DITA2Go** 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 §32.1 [Locating graphics files for HTML](#) on page 611.

*In this section:*

§44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796

§44.7.2 [Selecting graphics to copy from arbitrary locations](#) on page 797

§44.7.3 [Deleting prior contents of the graphics destination directory](#) on page 798

§44.7.4 [Synchronizing graphics settings for HTML output](#) on page 799

§44.7.5 [Synchronizing graphics settings for RTF output](#) on page 800

*See also:*

§44.6 [Assembling files for distribution](#) on page 792.

### 44.7.1 Copying referenced graphics to a distribution directory

When you specify `WrapAndShip=Yes` or designate a `WrapPath` directory, for HTML output **DITA2Go** automatically copies graphics files from the project directory to the directory designated by `[Graphics]GraphPath`; see §32.1 [Locating graphics files for HTML](#) on page 611. **DITA2Go** can also copy graphics files from other locations.

To have **DITA2Go** 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`, **DITA2Go** copies graphics from wherever they are referenced by your DITA document to one of the following destinations:

- for JavaHelp and Oracle Help for Java, the directory designated by `[JavaHelpOptions]GraphSubdir` (see §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389)
- for other HTML output types (and DCL output), the directory designated by `[Graphics]GraphPath`, if any (see §44.7.4 [Synchronizing graphics settings for HTML output](#) on page 799), otherwise the `WrapPath` directory
- for RTF output types (and MIF output), the directory designated by `WrapPath`.

In other words, **DITA2Go** 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 DITA 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 §40.2.1.2 [Substituting graphics files for HTML](#) on page 747.

**Note:** If you specify `[Automation]OnlyAuto=Yes` (see §44.13 [Postprocessing separately from converting](#) on page 807), 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`.

## 44.7.2 Selecting graphics to copy from arbitrary locations

In addition to (or instead of) having **DITA2Go** gather up copies of the graphics files referenced by your DITA document (see §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796), you can have **DITA2Go** copy all or selected graphics files from other locations.

The paths in your DITA 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 `FTSCommand=path\to\indexer` (see §44.10 [Gathering and processing Help-system files](#) on page 802).

*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 §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796. 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 §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796), 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 44-3](#) 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 44-3 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 **DITA2Go** 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, **DITA2Go** changes them to forward slashes before inserting references in HTML output, from HTML files to image files. See §32.1 [Locating graphics files for HTML](#) on page 611.

*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:

§44.7.4 [Synchronizing graphics settings for HTML output](#) on page 799

§44.7.5 [Synchronizing graphics settings for RTF output](#) on page 800

*Use system commands instead*

As an alternative, you can collect graphics from multiple locations with a series of system commands in a **DITA2Go** 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 **DITA2Go** macros, where backslash is used as an escape character), and the quotes around paths that includes spaces; see §43.1.5 [Supplying system commands in a macro](#) on page 779.

### 44.7.3 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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

EmptyGraphPath takes effect when WrapAndShip=Yes, CompileHelp=Yes, or FTSCCommand=path\to\indexer (see §44.10 [Gathering and processing Help-system files](#) on page 802).

When EmptyGraphPath=Yes, provided [Graphics]GraphPath does *not* point to the project directory, **DITA2Go** deletes the entire contents of the GraphPath directory before copying files into it. However, if either of the following is true, **DITA2Go** 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, **DITA2Go** does not delete files unless both EmptyWrapPath and EmptyGraphPath are set to Yes; see §44.6

[Assembling files for distribution](#) on page 792.

When EmptyGraphPath=No (the default), **DITA2Go** leaves the prior contents of the GraphPath directory in place.

#### 44.7.4 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 §20.3.6.2 [Letting DITA2Go set up the directory structure and copy files](#) on page 389.

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 <img> tags in your HTML output, as references to graphics files on the server; see §32.1 [Locating graphics files for HTML](#) on page 611. If your configuration file does not include a setting for GraphPath, by default <img> tags do not include a path, unless you specify a path in [GraphFiles] (see §40.2.1.2 [Substituting graphics files for HTML](#) on page 747).

If WrapPath points to the project directory:

- Set StripGraphPath=Yes
- Remove or comment out any setting for GraphPath
- Set GraphPathOverrides=No (see §40.2.1.3 [Overriding path specifications for referenced graphics](#) on page 748)
- 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:

§32.1 [Locating graphics files for HTML](#) on page 611

§40.2 [Replacing and relocating graphics files](#) on page 746

§42.2.9.4 [Overriding graphic properties for HTML](#) on page 774

## 44.7.5 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 40-1 [RTF replacement graphics file mappings and locations](#) on page 749.

See also:

§40.2 [Replacing and relocating graphics files](#) on page 746.

## 44.8 Placing CSS or XSL files for assembly

For HTML output types, when you specify WrapAndShip=Yes and designate a WrapPath directory, **DITA2Go** automatically copies CSS or XSL files from the project directory to the directory designated by [CSS]CssPath. **DITA2Go** can also automatically copy CSS or XSL files from another directory you specify.

To have **DITA2Go** 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 §44.10 [Gathering and processing Help-system files](#) on page 802).

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 §31.4.2 [Designating and locating a CSS file](#) on page 595); otherwise,
- the directory designated by WrapPath (see §44.6 [Assembling files for distribution](#) on page 792) 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 **DITA2Go** 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
```

## 44.9 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>DITA files, projects file</i>
D:\MyDoc\Graphics	<i>Graphics</i>
D:\MyDoc\HTML	<i><b>DITA2Go</b> 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 DITA files) would specify `D:\MyDoc\HTML` as the path for **DITA2Go** 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</p> <p>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</p> <p>Where to find the CSS files for this project. Path separators are <i>backslashes</i>.</p> <p>CopyGraphicsFrom=D:\MyDoc\Graphics</p> <p>Where to find graphics for this project. Path separators are <i>backslashes</i>.</p> <p>GraphCopyFiles=*.jpg *.gif</p> <p>Files you want from the CopyGraphicsFrom directory.</p>
[CSS]	<p>CssPath=. \styles</p> <p>Where CSS files should be relative to the HTML files that use them (that is, relative to the WrapPath directory). <b>DITA2Go</b> converts backslashes to forward slashes before writing these references in the HTML files.</p>
[Graphics]	<p>GraphPath=. \images</p> <p>Where the graphics should be relative to the HTML files that reference them (that is, relative to the WrapPath directory). <b>DITA2Go</b> converts backslashes to forward slashes before writing these references in the HTML files.</p>

## 44.10 Gathering and processing Help-system files

Most Help systems require additional steps after **DITA2Go** generates output files from your DITA document, and before archiving files for distribution. You can have **DITA2Go** automatically do the following:

WinHelp	Run the WinHelp compiler; see §17.2.10 <a href="#">Compiling a WinHelp project</a> on page 285.
HTML Help	Run the HTML Help compiler; see §18.13 <a href="#">Compiling and testing HTML Help</a> on page 346.
OmniHelp	Copy viewer files to the WrapPath directory (needed only if they are not already in the project directory); see §19.13 <a href="#">Assembling OmniHelp files for viewing</a> on page 380.
JavaHelp	Run the full-text-search indexing program; see §20.5.2 <a href="#">Creating a search index for JavaHelp</a> on page 398.
Oracle Help for Java	Run the full-text-search indexing program (although you might not get a usable search index); see §20.5.3 <a href="#">Creating a search index for Oracle Help</a> on page 399.
Eclipse Help	Archive topic files into doc.zip; see §21.8 <a href="#">Packaging Eclipse Help files</a> on page 427.

*Compile WinHelp or HTML Help*

To direct **DITA2Go** 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:

§17.2.6 [Setting basic WinHelp options in the configuration file](#) on page 284

§18.13 [Compiling and testing HTML Help](#) on page 346

*Index JavaHelp or  
Oracle Help*

To direct **DITA2Go** 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, **DITA2Go** does not run the indexer. See:

§20.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 397

*Certain  
automation  
settings are  
activated*

When CompileHelp=Yes or FTSCommand=path/to/indexer, **DITA2Go** acts on those [Automation] settings that need to be processed prior to compilation or indexing, *regardless of the setting for WrapAndShip*. Then **DITA2Go** runs the appropriate compiler or indexer. [Table 44-4](#) shows which settings are activated.

**Table 44-4 Automation settings activated by CompileHelp or FTSCommand**

[Automation] setting	Action	Ref.
CopyCssFrom	Copy CSS files from the designated directory	<a href="#">44.8</a>
CopyGraphicsFrom	Copy graphics files from the designated directory	<a href="#">44.7.1</a>
CssCopyFiles	Select only specified CSS files for copying	<a href="#">44.8</a>
EmptyGraphPath	Delete prior copied graphics files before copying	<a href="#">44.7.3</a>
EmptyOutputDir	Delete files from the project directory before conversion	<a href="#">44.4.1</a>
EmptyOutputFiles	Select only specified files to delete from the project directory	<a href="#">44.4.2</a>
EmptyWrapPath	Delete all files from the WrapPath directory before copying	<a href="#">44.6</a>
GraphCopyFiles	Select only specified graphics files for copying	<a href="#">44.7.1</a>
WrapCopyFiles	Copy only specified files from the project directory	<a href="#">44.6</a>
WrapPath	Directory to which files are copied for compiling and assembling for distribution	<a href="#">44.6</a>
ShipPath	Directory to which compiled or archived files are copied or moved	<a href="#">44.12</a>

*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 §44.2 [Activating and logging production of deliverables](#) on page 788. 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.

## 44.11 Archiving deliverables

To archive output files assembled for distribution, **DITA2Go** can automatically run a command-line archiving program such as `pkzip.exe`, or WinZip command-line add-on `wzip.exe`. **DITA2Go** 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...]
```

DITA2Go uses the following settings to put together the components of this command:

Component	DITA2Go archive setting(s)	Reference
<b>wzip</b>	ArchiveCommand	<a href="#">44.11.1</a>
[options]	ArchiveStartParams	<a href="#">44.11.2</a>
zipfile	ArchiveName, ArchiveVer, ArchiveExt	<a href="#">44.11.3</a>
[files...]	ArchiveEndParams	<a href="#">44.11.2</a>

In this section:

[§44.11.1 Specifying an archiving command](#) on page 804

[§44.11.2 Supplying parameters for the archiving command](#) on page 804

[§44.11.3 Specifying archive file name and optional version](#) on page 805

## 44.11.1 Specifying an archiving command

To have DITA2Go 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 **DITA2Go User's Guide** is:

```
ArchiveCommand = "g:\program files\winzip\wzip"
```

ArchiveCommand has no default value; if you do not specify a value, DITA2Go 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 [§44.2 Activating and logging production of deliverables](#) on page 788)
- OnlyAuto=Yes (see [§44.13 Postprocessing separately from converting](#) on page 807).

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 [§44.11.2 Supplying parameters for the archiving command](#) on page 804.

## 44.11.2 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 **DITA2Go 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 **DITA2Go 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 §44.6 [Assembling files for distribution](#) on page 792), 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 **DITA2Go User's Guide**, which does not use a WrapPath directory, the setting specifies each file to be included:

```
ArchiveEndParams = ugdita2go.chm
```

### 44.11.3 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 **DITA2Go 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 base name of the map file. [Table 44-5](#) shows the source of the default base file name of the archive for each output type.



**Table 44-5 Default base file name for deliverables archive**

Output type	Source of default base file name for archive	Ref.
HTML Help	[MSHtmlHelpOptions]HHPFileName	18.3.6
JavaHelp, Oracle Help	[JavaHelpOptions]HSFileName	20.3.7
OmniHelp	[OmniHelpOptions]ProjectName ( <i>without prefix or suffix</i> )	19.3.2
WinHelp	[HelpOptions]HPJFileName	17.2.6
Eclipse Help	plugin ( <i>literally</i> )	21.8.3
All other output types	Base name of project map file	2.3.5
All other output types	Base name of the map	

*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 26. If you do not specify a value for ArchiveVer, **DITA2Go** 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 §44.11.1 [Specifying an archiving command](#) on page 804). If ArchiveCommand contains **jar**, the default extension is .jar; otherwise the default extension is .zip. **DITA2Go** provides the leading period.

## 44.12 Placing deliverables in a shipping directory

You can have **DITA2Go** copy or move deliverable files to a separate directory for shipping, or sharing, or storage.

*In this section:*

§44.12.1 [Specifying a shipping directory for deliverables](#) on page 806

§44.12.2 [Understanding which files are placed in the shipping directory](#) on page 807

§44.12.3 [Choosing whether to copy or move deliverables](#) on page 807

### 44.12.1 Specifying a shipping directory for deliverables

To have **DITA2Go** 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 §44.2 [Activating and logging production of deliverables](#) on page 788.

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, **DITA2Go** creates this directory for you.

## 44.12.2 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 **DITA2Go** project.

If you specify a value for `ArchiveCommand` (see §44.11 [Archiving deliverables](#) on page 803), **DITA2Go** 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>

## 44.12.3 Choosing whether to copy or move deliverables

When `WrapAndShip=Yes` and a value is specified for `ShipPath`, by default **DITA2Go** copies deliverables to the `ShipPath` directory, leaving the originals in the `WrapPath` directory.

To have **DITA2Go** 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 §44.2 [Activating and logging production of deliverables](#) on page 788) and `ShipPath` has a non-blank value.

## 44.13 Postprocessing separately from converting

If you have already converted a document and the results are still in the project directory, you can have **DITA2Go** 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 §44.2 [Activating and logging production of deliverables](#) on page 788) *except* `CopyOriginalGraphics`; see §44.7.1 [Copying referenced graphics to a distribution directory](#) on page 796.

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, **DITA2Go** 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 §43.1 [Executing operating-system commands](#) on page 777.

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, **DITA2Go** 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), **DITA2Go** runs the appropriate compiler before placing the deliverable(s) in a shipping directory; see §44.10 [Gathering and processing Help-system files](#) on page 802.

**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, **DITA2Go** runs the designated indexer before archiving the deliverables and placing them in a shipping directory; see §20.5 [Providing full-text search for JavaHelp / Oracle Help](#) on page 397.

# 45 Converting via DCL

---

This section shows how to operate the **DITA2Go** DCL filter. Topics include:

- §45.1 [How the DCL filter works](#) on page 809
- §45.2 [Using the DCL filter](#) on page 809
- §45.3 [DCL command-line syntax](#) on page 810
- §45.4 [Specifying output file paths and names](#) on page 812
- §45.5 [About DCL technology](#) on page 813

## 45.1 How the DCL filter works

**DITA2Go** uses the DCL (Document Coding Language) filter to convert XML 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 set up a configuration file for the conversion; see §2 [Converting DITA documents](#) on page 39.

When your configuration file is ready, you run the **DITA2Go** DCL filter at a command-line prompt in a command window.

You will also need to use the full path to the **DITA2Go** version of `dcl.exe` on the command line; see §2.7.1 [Executing the correct version of DCL](#) on page 46.

## 45.2 Using the DCL filter

*In this section:*

- §45.2.1 [Understanding where to run DCL](#) on page 809
- §45.2.2 [Preparing for conversion](#) on page 809
- §45.2.3 [Converting a single DITA or DCL file](#) on page 809
- §45.2.4 [Converting a group of files](#) on page 810

### 45.2.1 Understanding where to run DCL

You must invoke the **DITA2Go** DCL filter on a command line in a Windows *Command Prompt* window. The **DITA2Go** DCL filter is a Windows Console application, not an MS-DOS application. It will not run under plain MS-DOS, without Windows.

### 45.2.2 Preparing for conversion

Before you use the DCL command-line method to convert files, you must do the following:

1. Copy a starting configuration file for the output type you want (see [Table 39-5](#) on page 737) from `%OMSYSHOME%\d2g\local\config` to your output directory.
2. Edit the configuration file to specify settings. See §3.1 [Working with DITA2Go configuration files](#) on page 49 for more information.

### 45.2.3 Converting a single DITA or DCL file

To convert a single DITA XML file with the **DITA2Go** DCL filter:

1. Open a Windows *Command Prompt* window.

2. Change to the directory where you placed a configuration file.
3. At the command-line prompt, enter the following command:

```
dcl -f format [-o output] input
```

where the arguments are as follows:

<i>format</i>	Output type; one of the format codes or names listed for <b>-f</b> in §45.3.1 <a href="#">Command-line switch -f format</a> on page 810.
<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.

## 45.2.4 Converting a group of files

You can convert more than one file at a time, by using wildcards in file names, or by executing the **DITA2Go** DCL filter in a batch file. You must make sure any path values in the [Setup] section of the configuration file are correct.

## 45.3 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:*

- §45.3.1 [Command-line switch -f format](#) on page 810
- §45.3.2 [Command-line switch -o output](#) on page 811
- §45.3.3 [Command-line argument input ...](#) on page 812
- §45.3.4 [Command-line switch -v](#) on page 812
- §45.3.5 [Additional command-line switches](#) on page 812

### 45.3.1 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		HTML 4.0
XHTML		XHTML 1.0
HTMLHelp	C I B	Microsoft HTML Help
JavaHelp	C I B	JavaHelp
OracleHelp	C I B	Oracle Help for Java
EclipseHelp	C I B	Eclipse Help

<u>Format name</u>	<u>Optional suffix</u>	<u>Description</u>
OmniHelp	C I B	Cross-platform OmniHelp
DITA		DITA XML
DocBook		DocBook XML
XML		Generic XML
Word		Word 8/97
WinHelp		WinHelp 4

*Merge contents  
and/or index*

Where listed under **Suffix**, you can append one of the following letters to the format name (case does not matter). These suffix options apply to HTML-based Help systems for which **DITA2Go** 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 §45.3.3 [Command-line argument input ...](#) on page 812.

### 45.3.2 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 §45.3.5 [Additional command-line switches](#) on page 812), the default value for *.ext* depends on the value of the **-f** argument (see §45.3.1 [Command-line switch -f format](#) on page 810). 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**.

### 45.3.3 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.dita"
```

When you specify suffix C, I, or B for the **-f** switch, the input file must have extension .lst; see §45.3.1 [Command-line switch -f format](#) on page 810.

### 45.3.4 Command-line switch -v

The **-v** switch produces verbose output; **dcl** reports, at the command prompt, everything it does.

### 45.3.5 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 45-1](#) shows the options for these additional switches.

**Table 45-1 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 §45.5.1 <a href="#">DCL file structure</a> on page 813
dcl	Intermediate Document Coding Language ASCII format; see §45.5.1 <a href="#">DCL file structure</a> on page 813
htm	HTML, HTML-based Help, XML
rtf	Rich Text Format
XML	DITA, DocBook, generic XML

## 45.4 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, and if the file naming rules for the system permit multiple extensions.)

## 45.5 About DCL technology

The **DITA2Go** 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 **DITA2Go**, are proprietary, and must be licensed from Omni Systems.

*In this section:*

§45.5.1 [DCL file structure](#) on page 813

§45.5.2 [Writing DCL conversion modules](#) on page 813

### 45.5.1 DCL file structure

DCL can be read and written in either of two formats: ASCII or binary. When the **DITA2Go** 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, `drmf` can write either format, and `dwrftf` can read either format.

### 45.5.2 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.



# 46 Creating a map with DITA2Map

---

**DITA2Map** is a command-line program that generates a map from a DITA topic file, “a nice little free utility to help people mired in the nested-topics world into the cleaner world of maps”. Topics include:

- §46.1 [Understanding how DITA2Map works](#) on page 815
- §46.2 [Setting up a DITA2Map project](#) on page 815
- §46.3 [Specifying DITA2Map configuration options](#) on page 815
- §46.4 [Running DITA2Map](#) on page 817

## 46.1 Understanding how DITA2Map works

If you have a DITA topic file, either `.dita` or one of its specializations, or a collection with `<dita>` as root, you can use command-line program **DITA2Map** to generate a `.ditamap`.

Working from an optional configuration file, a DTD, and your DITA topic file, **DITA2Map** generates a `.ditamap` by making a topicref from each topic ID, title, and shortdesc (if any) in the topic file. **DITA2Map** nests the topicrefs as the topics are nested, and optionally adds a reltable with one row and a column for each topic type found.

## 46.2 Setting up a DITA2Map project

**DITA2Map** requires DITA DTDs, and executables `dita2map.exe` and `libexpat.dll`. If you have installed **DITA2Go** you already have the DTDs and these executable files.

Unless **DITA2Go** configuration file `dita2map.ini` is already present in the directory where you want **DITA2Map** to create a map, you might want to create a configuration file named `dita2map.ini`, and populate it with the options listed in §46.3 [Specifying DITA2Map configuration options](#) on page 815. However, a configuration file is optional; instead, you can accept the defaults for most configuration settings, and specify the rest on the command line. To determine processing options, **DITA2Map** looks first for `dita2map.ini`; if that file is not present, **DITA2Map** uses default settings.

To use **DITA2Map**, you must have at least the demonstration version of **DITA2Go** installed on your system.

## 46.3 Specifying DITA2Map configuration options

Edit file `dita2map.ini` to specify options for creating a map file from a DITA topic file. Or, edit the same options in your project configuration file. Or, accept default values and dispense with a configuration file.

*In this section:*

- §46.3.1 [Locating a DITA DTD](#) on page 816
- §46.3.2 [Locating configuration template files](#) on page 816
- §46.3.3 [Specifying processing options](#) on page 816
- §46.3.4 [Specifying logging options](#) on page 816
- §46.3.5 [Specifying map options for DITA2Map](#) on page 817

### 46.3.1 Locating a DITA DTD

To specify the location of DITA DTD files:

```
[Setup]
; DTDPPath = path to location of DITA DTDs, default none
DTDPPath=D:\path\to\DTDs
```

The OASIS DITA 1.1 DTD is in directory %omshome%\d2g\DTD of your **DITA2Go** installation.

You can override the setting for DTDPPath with the **-d** switch on the dita2map.exe command line; see §46.4 [Running DITA2Map](#) on page 817.

If you run **DITA2Map** without specifying a value for DTDPPath, either in a configuration file or with the **-d** switch, **DITA2Map** will try to read the DTD referenced in the SYSTEM ID, normally the one at OASIS, and most likely will fail. That is, **DITA2Map** will write an empty map with no topicrefs, just header and ending.

### 46.3.2 Locating configuration template files

If you expect to need multiple configuration files for different **DITA2Map** projects, you can collect settings that are common to all projects in a configuration template; see §39.2 [Referencing configuration files and templates](#) on page 731.

To specify a configuration template:

```
[Templates]
; Configs = path to .ini file
Configs=D:\path\to\template.ini
```

**DITA2Map** checks the template file for any settings missing from dita2map.ini.

### 46.3.3 Specifying processing options

To determine how names are matched to configuration settings:

```
[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
```

See §4.1.10 [Specifying how to treat cases, spaces, and wildcards](#) on page 73.

To diagnose possible memory-management problems:

```
[Options]
NoNameDel = No
NoMemDel = No
```

If dcl.exe crashes at the very end of processing, try changing these options to Yes, one at a time.

### 46.3.4 Specifying logging options

Logging options for **DITA2Map** are the same as for **DITA2Go**, except for the default log file name:

```
[Logging]
; UseLog = Yes (default, log as specified in this section) or No
UseLog = Yes
```

```
; LogFileName = name with path, absolute or relative to output dir
LogFileName = DITA2map_log.txt
```

See §4.2 [Logging conversion events](#) on page 74.

### 46.3.5 Specifying map options for DITA2Map

**DITA2Map** uses the same map-generation options as **DITA2Go**. If you are happy with the default settings listed in §4.1.4.2 [Specifying options for a generated map](#) on page 69, except for the title of the map, you can specify the map title on the **DITA2Map** command line. See §46.4 [Running DITA2Map](#) on page 817.

## 46.4 Running DITA2Map

You must run **DITA2Map** from a command line, in a Windows *Command Prompt* window. **DITA2Map** looks for a configuration file in the current directory. However, a configuration file is optional. If you are satisfied with default values for most configuration settings, you can specify the rest with command-line switches.

To use `dita2map.exe` to generate a `.ditamap` file from a DITA topic file:

1. Open a Windows *Command Prompt* window.
2. Navigate to the directory where you want to use the map file that **DITA2Map** produces, and where an optional **DITA2Map** configuration file (`dita2map.ini`) or **DITA2Go** project configuration file is located.
3. Type a command of the following form (*all on one line*):

```
dita2map [-v] [-t "Map Title"] [-d "D:\path\to\DTDs"]
        [-i inifile.ini] [-o mapfile.ditamap] topicfile.dita
```

where the switches specify the following:

- v    Verbose command-line output.
- t    Content of the `<title>` element for the map, enclosed in double quotes.
- d    Relative or absolute path to local DITA DTDs, enclosed in double quotes.
- i    Name of a configuration file to use; must be in the current directory.
- o    Name of the map file to produce.

4. Press **Enter**.

All the command-line arguments to `dita2map.exe` are optional except the name of the DITA topic file from which to generate the map. You can include a path for `topicfile`. The generated topicrefs use the name of the topic file exactly as you specify it on the command line, including any path. Links in the topicrefs are all relative to the directory where **DITA2Map** is run. This is generally not the same directory where you run **DITA2Go**, because that is the output directory, which is usually different from the input directory. This means you should run `dita2map.exe` from the **DITA2Go** directory, which is where you want to use the `.ditamap` file **DITA2Map** produces.

If you run `dita2map.exe` without the `-o` switch, the map file will have the same name as the input topic file, except with extension `.ditamap`.

By default, `dita2map.exe` produces a log file called `DITA2map_log.txt`; see §46.3.4 [Specifying logging options](#) on page 816.



# A Technical support for DITA2Go

---

Omni Systems can provide effective technical support for **DITA2Go** 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:

§A.1 [Things to check first](#) on page 819

§A.2 [How to request help](#) on page 821

**Zip your files!** *Do not send unzipped DITA files to Omni Systems.  
Do not send files larger than 1 MB.*

## A.1 Things to check first

Before you holler for help, try the following:

§A.1.1 [Examine your conversion log file](#) on page 819

§A.1.2 [Check your DITA2Go installation](#) on page 819

§A.1.3 [Check the DITA2Go User's Guide](#) on page 819

§A.1.4 [Check path names, file names, and drive location](#) on page 820

§A.1.5 [Check your version of DITA2Go](#) on page 820

### A.1.1 Examine your conversion log file

By default, **DITA2Go** 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 §4.2 [Logging conversion events](#) on page 74.

*No log errors or warnings?* Next: [Check your DITA2Go installation](#)

### A.1.2 Check your DITA2Go installation

Sometimes supporting files are not where they need to be.

If you use **DITA2Go** 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 §19.2 [Setting up OmniHelp viewer control files](#) on page 354.

*Everything where it belongs?* Next: [Check the DITA2Go User's Guide](#).

### A.1.3 Check the DITA2Go User's Guide

Quickest way: use the **Search** feature of the HTML Help version, which is installed with **DITA2Go**. 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](#).



## A.1.4 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 26.

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. A network is inherently slower and less reliable than a local system, so you might have mysterious and intermittent problems when a heavily loaded network is used for file storage.

*File names and paths valid, and the problem still exists?* Next: [Check your version of DITA2Go](#).

## A.1.5 Check your version of DITA2Go

If you are using **DITA2Go**, do the following:

1. In a text editor, open an output file (.htm or .rtf or .dita or .ent) that you created with **DITA2Go**, and find a line near the top of the file that shows the **DITA2Go** version and build numbers. The line you want looks like this:

```
HTML/XML: <!-- generated by DCL filter dwhtml, Ver 4.0 x002 h289 -->
RTF:      {\info {\doccomm DCL filter dwrtf, Ver 4.0 x002 r297}}
```

[Table A-1](#) on page 820 lists the build numbers underlined in these examples.

2. Go to the **DITA2Go** Web site:

<http://www.dita2go.com/>

Navigate to **Downloads > Basic 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>
drxml <u>002</u> .zip	215k	DITA input module	01-Jun-2010
dwrftf <u>295</u> .zip	251k	RTF output module	01-Jun-2010
dwhtml <u>289</u> .zip	421k	HTML/XML output module	01-Jun-2010

3. Compare the number on each DLL archive file with the build numbers you found in your output, as shown in [Table A-1](#).

**Table A-1 Examples of build numbers for DITA2Go DLL files**

Output type	DLL file	Build number:		
		Latest	Used	Current?
All	drxml.dll	002	x002	Yes
HTML	dwhtml.dll	289	h284	No
RTF	dwrftf.dll	295	r295	Yes

4. If the build number on a DLL archive is higher than the corresponding build number in your output file, obtain and install the current update; see §1.4 [How to update DITA2Go](#) on page 36. Then try the conversion again.
5. 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.
6. If you still encounter the problem, check whether later beta versions of any DLLs are available on the **DITA2Go** Web site:

<http://www.dita2go.com/>

Navigate to **Downloads > Basic 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>
drxml.dll	580k	DITA input module, 4.0.3.42	04-Jun-2012
dwrftf.dll	588k	RTF output module, 4.0.297.2	02-Jun-2012
dwhtml.dll	1052k	HTML/XML output module, 4.0.291.4	04-Jun-2012

7. Compare the third part of the number in each description with the build numbers you found in your output, as shown in [Table A-1](#). If the fourth part is greater than zero, and the problem is due to a defect in **DITA2Go**, the defect might have been corrected. See §1.4.2 [Try out DITA2Go beta executables](#) on page 37.

*Still no luck?* See §A.2 [How to request help](#) on page 821.

## A.2 How to request help

*Zip your files!* **Do not send unzipped DITA files to Omni Systems.**  
**Do not send files larger than 1 MB.**

If you still encounter problems after following the steps in §A.1 [Things to check first](#) on page 819, help us to help you, as follows:

- §A.2.1 [If the problem involves a crash](#) on page 821
- §A.2.2 [Scope the problem](#) on page 822
- §A.2.3 [Document the problem](#) on page 822
- §A.2.4 [Package the problem](#) on page 822
- §A.2.5 [Send the package to Omni Systems](#) on page 823

### A.2.1 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 **DITA2Go** conversion caused a crash. 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 **DITA2Go** is always freed; no memory leaks occur.

Next: [Scope the problem](#).

## A.2.2 Scope the problem

Use the Configuration Manager (see §3.2 [Editing files with the Configuration Manager](#) on page 49) to check the configurations in use. If any settings in ‘local’ configuration files in %OMSYSHOME%\d2g\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](#).

## A.2.3 Document the problem

Write an e-mail message that contains the following information:

- A brief description of the problem, including answers to questions in §A.2.2 [Scope the problem](#) on page 822.
- Operating-system name and version; for example, Windows 7 X64.
- Amount of memory on your machine; for example, 2 GB.
- Browser name and version, if the problem occurs when you generate HTML; for example, Firefox 3.5.

Next: [Package the problem](#).

## A.2.4 Package the problem

If you have placed settings in a file in %OMSYSHOME%\d2g\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:

DITA file(s)	The smallest fragments that yield the problem. <b><i>No unzipped files.</i></b>
Output file(s)	Whatever output (if any) shows the undesired result.
Log file	Located by default in your project directory; see §4.2 <a href="#">Logging conversion events</a> on page 74.
Configuration file(s)	Your project configuration file, plus any chapter-specific configuration file used by the problem DITA 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%\d2g\local\config\*, copy those settings to your project configuration file before you create the package.
Macro libraries	If the problem file uses <b>DITA2Go</b> macros located in library files.
CSS file(s)	If your project uses CSS (HTML output only), and the problem is a display problem.

Help project file	If the problem occurs when you generate one of the following: <ul style="list-style-type: none"><li>• WinHelp: <i>MyDoc</i> .hlpj.</li><li>• HTML Help: <i>MyDoc</i> .hhp.</li><li>• JavaHelp or Oracle Help for Java: <i>MyDoc</i> .hs.</li></ul>
Graphics files	Any external graphics referenced by the problem-file fragment.

***Zip your files! Do not send unzipped files.***

Finally: [Send the package to Omni Systems.](#)

## A.2.5 Send the package to Omni Systems

Attach the .zip file you created in §A.2.4 [Package the problem](#) on page 822 to the e-mail message you wrote in §A.2.3 [Document the problem](#) on page 822, and send it to:

[support@omsys.com](mailto: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 DITA files to Omni Systems.  
Do not send files larger than 1 MB.***



# B Element type default properties

Table B-1 shows the default properties DITA2Go assigns to the elements shown in the first column, qualified by their class attribute values, shown in the second column. See §11 [Defining element sets and properties](#) on page 179.

**Table B-1** Default properties assigned to elements

Element name	Class attribute	Default element type properties
abbreviated-form	topic/term abbrev-d/abbreviated-form	Inline Gloss Abbrev
abbrevlist	map/topicref bookmap/abbrevlist	Map Ref Topic List
abstract	topic/abstract	Text Abstr
alt	topic/alt	Text Inline Alt
amendments	map/topicref bookmap/amendments	Map Ref Topic List
anchor	map/anchor	
anchorref	map/topicref mapgroup-d/anchorref	Map Ref Topic
apiname	topic/keyword pr-d/apiname	Text Inline
appendices	map/topicref bookmap/appendices	Map Ref Topic
appendix	map/topicref bookmap/appendix	Map Ref Topic
approved	topic/data bookmap/approved	
area	topic/figgroup ut-d/area	Inline Image Ref
audience	topic/audience	
author	topic/author	Text Var
b	topic/ph hi-d/b	Text Inline Typo
backmatter	map/topicref bookmap/backmatter	Map List
bibliolist	map/topicref bookmap/bibliolist	Map Ref Topic List
body	topic/body	
bodydiv	topic/bodydiv	Text Section
bookabstract	map/topicref bookmap/bookabstract	Topic Map Ref
bookchangehistory	topic/data bookmap/bookchangehistory	
bookevent	topic/data bookmap/bookevent	
bookeventtype	topic/data bookmap/bookeventtype	
bookid	topic/data bookmap/bookid	Map List
booklibrary	topic/ph bookmap/booklibrary	Text Inline Map
booklist	map/topicref bookmap/booklist	Map Ref Topic List
booklists	map/topicref bookmap/booklists	Map List
bookmap	map/map bookmap/bookmap	Root Map
bookmeta	map/topicmeta bookmap/bookmeta	Map Meta Topic
booknumber	topic/data bookmap/booknumber	Text Map Var
bookowner	topic/data bookmap/bookowner	Text Map Var
bookpartno	topic/data bookmap/bookpartno	Text Map Var
bookrestriction	topic/data bookmap/bookrestriction	Text Map Var
bookrights	topic/data bookmap/bookrights	Text Map Var
booktitle	topic/title bookmap/booktitle	Map
booktitlealt	topic/ph bookmap/booktitlealt	Text Map Var
boolean	topic/boolean	Inline

**Table B-1 Default properties assigned to elements (continued)**

Element name	Class attribute	Default element type properties
brand	topic/ <b>brand</b>	Text Var
category	topic/ <b>category</b>	Text Var
chapter	map/topicref bookmap/ <b>chapter</b>	Map Ref Topic
chdesc	topic/stentry task/ <b>chdesc</b>	Task Text TabCell
chdeschd	topic/stentry task/ <b>chdeschd</b>	Task Text TabCell
chhead	topic/sthead task/ <b>chhead</b>	Task TabRow TabHead
choice	topic/li task/ <b>choice</b>	Task Text ListItem XRSOURCE Pernicious
choices	topic/ul task/ <b>choices</b>	Task List
choicetable	topic/simpletable task/ <b>choicetable</b>	Task Table TabStart
choption	topic/stentry task/ <b>choption</b>	Task Text TabCell
choptionhd	topic/stentry task/ <b>choptionhd</b>	Task Text TabCell
chrow	topic/strow task/ <b>chrow</b>	Task TabRow TabBody
cite	topic/ <b>cite</b>	Text Inline
cmd	topic/ph task/ <b>cmd</b>	Task Text Inline
cmdname	topic/keyword sw-d/ <b>cmdname</b>	Text Inline
codeblock	topic/pre pr-d/ <b>codeblock</b>	Text Pre
codeph	topic/ph pr-d/ <b>codeph</b>	Text Inline Pre
coderef	topic/xref pr-d/ <b>coderef</b>	Text Pre Ref
colophon	map/topicref bookmap/ <b>colophon</b>	Map Ref Topic
colspec	topic/ <b>colspec</b>	Table TabCol
completed	topic/ph bookmap/ <b>completed</b>	Text Map Inline
component	topic/ <b>component</b>	Text Var
conbody	topic/body concept/ <b>conbody</b>	
conbodydiv	topic/bodydiv concept/ <b>conbodydiv</b>	Text Section
concept	topic/topic concept/ <b>concept</b>	Root
context	topic/section task/ <b>context</b>	Task Text Section
coords	topic/ph ut-d/ <b>coords</b>	Text Image Group
copyrfirst	topic/data bookmap/ <b>copyrfirst</b>	
copyrholder	topic/ <b>copyrholder</b>	Text Var
copyright	topic/ <b>copyright</b>	
copyrlast	topic/data bookmap/ <b>copyrlast</b>	
copyryear	topic/ <b>copyryear</b>	
created	topic/ <b>created</b>	
critdates	topic/ <b>critdates</b>	
data	topic/ <b>data</b>	Data Text Inline
data-about	topic/ <b>data-about</b>	Data Inline
day	topic/ph bookmap/ <b>day</b>	Text Map Inline
dd	topic/ <b>dd</b>	Text TabCell DLDef
ddhd	topic/ <b>ddhd</b>	Text TabCell DLDef
dedication	map/topicref bookmap/ <b>dedication</b>	Map Ref Topic
delim	topic/ph pr-d/ <b>delim</b>	Text
desc	topic/ <b>desc</b>	Text Desc



**Table B-1 Default properties assigned to elements (continued)**

Element name	Class attribute	Default element type properties
dita	dita	
dl	topic/ <b>dl</b>	Table TabStart DList
dlentry	topic/ <b>dlentry</b>	TabRow TabBody DLEntry
dlhead	topic/ <b>dlhead</b>	TabRow TabHead DLEntry
draft-comment	topic/ <b>draft-comment</b>	Text Inline Draft Pernicious
draftintro	map/topicref bookmap/ <b>draftintro</b>	Map Ref Topic
dt	topic/ <b>dt</b>	Text TabCell DLTerm
dthd	topic/ <b>dthd</b>	Text TabCell DLTerm
edited	topic/data bookmap/ <b>edited</b>	
edition	topic/data bookmap/ <b>edition</b>	
entry	topic/ <b>entry</b>	Text TabCell Pernicious
example	topic/ <b>example</b>	Task Text Section
featnum	topic/ <b>featnum</b>	Text Var
fig	topic/ <b>fig</b>	Fig
figgroup	topic/ <b>figgroup</b>	
figurelist	map/topicref bookmap/ <b>figurelist</b>	Map Ref Topic List
filepath	topic/ph sw-d/ <b>filepath</b>	Text Inline
fn	topic/ <b>fn</b>	Text Inline Footnote Num XRSource
foreign	topic/ <b>foreign</b>	Inline
fragment	topic/figgroup pr-d/ <b>fragment</b>	
fragref	topic/xref pr-d/ <b>fragref</b>	Text
frontmatter	map/topicref bookmap/ <b>frontmatter</b>	Map List
glossarylist	map/topicref bookmap/ <b>glossarylist</b>	Map Ref Topic List Glossary
glossdef	topic/abstract concept/abstract glossentry/ <b>glossdef</b>	Text Abstr Glossary
glossentry	topic/topic concept/concept glossentry/ <b>glossentry</b>	Root Glossary
glossgroup	topic/topic concept/concept glossegroup <b>glossgroup</b>	Root
glossref	map/topicref glossref-d/glossref	Map Ref Topic Glossary
glossterm	topic/title concept/title glossentry/ <b>glossterm</b>	Text Title XRSource Glossary
groupchoice	topic/figgroup pr-d/ <b>groupchoice</b>	
groupcomp	topic/figgroup pr-d/ <b>groupcomp</b>	
groupseq	topic/figgroup pr-d/ <b>groupseq</b>	
i	topic/ph hi-d/ <b>i</b>	Text Inline Typo
image	topic/ <b>image</b>	Inline Image
imagemap	topic/fig ut-d/ <b>imagemap</b>	Fig Inline Image Ref Group
index-base	topic/ <b>index-base</b>	Text Inline Index
index-see	topic/index-base indexing-d/ <b>index-see</b>	Text Inline Index IxSee
index-see-also	topic/index-base indexing-d/ <b>index-see-also</b>	Text Inline Index IxSeeAlso
index-sort-as	topic/index-base indexing-d/ <b>index-sort-as</b>	Text Inline Index IxSort
indexlist	map/topicref bookmap/ <b>indexlist</b>	Map Ref Topic List

**Table B-1 Default properties assigned to elements (continued)**

Element name	Class attribute	Default element type properties
indexterm	topic/ <b>indexterm</b>	Text Inline IxStart
indextermref	topic/ <b>indextermref</b>	Inline Index
info	topic/itemgroup task/ <b>info</b>	Task Text Inline
isbn	topic/data bookmap/ <b>isbn</b>	Text Map Var
itemgroup	topic/ <b>itemgroup</b>	Text Inline
keydef	map/topicref mapgroup-d/ <b>keydef</b>	Key Map Ref Topic
keyword	topic/ <b>keyword</b>	Text Inline
keywords	topic/ <b>keywords</b>	
kwd	topic/keyword pr-d/ <b>kwd</b>	Text
li	topic/ <b>li</b>	Text ListItem XRSource Pernicious
lines	topic/ <b>lines</b>	Text Pre
link	topic/ <b>link</b>	Link
linkinfo	topic/ <b>linkinfo</b>	Link Text Sequence
linklist	topic/ <b>linklist</b>	Link List Sequence
linkpool	topic/ <b>linkpool</b>	Link List
linktext	map/ <b>linktext</b>	Link Text Map
linktext	topic/ <b>linktext</b>	Link Text
longdesc	topic/longdesc	Inline Ref
longquoderef	topic/longquoderef	Inline Ref
lq	topic/ <b>lq</b>	Text
mainbooktitle	topic/ph bookmap/ <b>mainbooktitle</b>	Text Map Var
maintainer	topic/data bookmap/ <b>maintainer</b>	
map	map/ <b>map</b>	Root Map
mapref	map/topicref mapgroup-d/ <b>mapref</b>	Map Ref Topic
menucascade	topic/ph ui-d/ <b>menucascade</b>	Inline CascadeSet
metadata	topic/ <b>metadata</b>	
month	topic/ph bookmap/ <b>month</b>	Text Map Inline
msgblock	topic/pre sw-d/ <b>msgblock</b>	Text Pre
msgnum	topic/keyword sw-d/ <b>msgnum</b>	Text Inline
msgph	topic/ph sw-d/ <b>msgph</b>	Text Inline
navref	map/ <b>navref</b>	
navtitle	topic/ <b>navtitle</b>	Text
no-topic-nesting	topic/ <b>no-topic-nesting</b>	
note	topic/ <b>note</b>	Text Note
notices	map/topicref bookmap/ <b>notices</b>	Map Ref Topic
object	topic/ <b>object</b>	Object
ol	topic/ <b>ol</b>	List Num
oper	topic/ph pr-d/ <b>oper</b>	Text
option	topic/keyword pr-d/ <b>option</b>	Text Inline
organization	topic/data bookmap/ <b>organization</b>	Map
othermeta	topic/ <b>othermeta</b>	
p	topic/ <b>p</b>	Text

**Table B-1 Default properties assigned to elements (continued)**

Element name	Class attribute	Default element type properties
param	topic/ <b>param</b>	Object Param
parml	topic/dl pr-d/ <b>parml</b>	Table TabStart PList
parmname	topic/keyword pr-d/ <b>parmname</b>	Text Inline
part	map/topicref bookmap/ <b>part</b>	Map Ref Topic
pd	topic/dd pr-d/ <b>pd</b>	Text TabCell PLDef
permissions	topic/ <b>permissions</b>	
person	topic/data bookmap/ <b>person</b>	Map
ph	topic/ <b>ph</b>	Text Inline
platform	topic/ <b>platform</b>	Text Var
plentry	topic/dlentry pr-d/ <b>plentry</b>	TabRow TabBody PLEntry
postreq	topic/section task/ <b>postreq</b>	Task Text Section
pre	topic/ <b>pre</b>	Text Pre
preface	map/topicref bookmap/ <b>preface</b>	Map Ref Topic
prereq	topic/section task/ <b>prereq</b>	Task Text Section
printlocation	topic/data bookmap/ <b>printlocation</b>	Map
prodinfo	topic/ <b>prodinfo</b>	
prodname	topic/ <b>prodname</b>	Text Var
prognum	topic/ <b>prognum</b>	Text Var
prolog	topic/ <b>prolog</b>	Meta Group
propdesc	topic/stentry reference/ <b>propdesc</b>	Text TabCell Reference
propdeschd	topic/stentry reference/ <b>propdeschd</b>	Text TabCell Reference
properties	topic/simpletable reference/ <b>properties</b>	Table TabStart Reference
property	topic/strow reference/ <b>property</b>	TabRow TabBody Reference
prophead	topic/sthead reference/ <b>prophead</b>	TabRow TabHead Reference
proptype	topic/stentry reference/ <b>proptype</b>	Text TabCell Reference
proptypehd	topic/stentry reference/ <b>proptypehd</b>	Text TabCell Reference
propvalue	topic/stentry reference/ <b>propvalue</b>	Text TabCell Reference
propvaluehd	topic/stentry reference/ <b>propvaluehd</b>	Text TabCell Reference
pt	topic/dt pr-d/ <b>pt</b>	Text TabCell PLTerm
published	topic/data bookmap/ <b>published</b>	Map
publisher	topic/ <b>publisher</b>	Text Var
publisherinformation	topic/publisher bookmap/ <b>publisherinformation</b>	Map Text Var
publishtype	topic/data bookmap/ <b>publishtype</b>	Map
q	topic/ <b>q</b>	Text Inline Typo
refbody	topic/body reference/ <b>refbody</b>	
refbodydiv	topic/bodydiv reference/ <b>refbodydiv</b>	Text Section Reference
reference	topic/topic reference/ <b>reference</b>	Root Reference
refsyn	topic/section reference/ <b>refsyn</b>	Text Section Reference
related-links	topic/ <b>related-links</b>	Rel
relcell	map/ <b>relcell</b>	TabCell
relcolspec	map/ <b>relcolspec</b>	Table TabCol
relheader	map/ <b>relheader</b>	TabRow TabHead

Table B-1 Default properties assigned to elements (continued)

Element name	Class attribute	Default element type properties
relrow	map/ <b>relrow</b>	TabRow TabBody
reltable	map/ <b>reltable</b>	Map Table TabStart Rel
repsep	topic/ph pr-d/ <b>repsep</b>	Text
required-cleanup	topic/ <b>required-cleanup</b>	Inline
resourceid	topic/ <b>resourceid</b>	
result	topic/section task/ <b>result</b>	Task Text Section
reviewed	topic/data bookmap/ <b>reviewed</b>	Map
revised	topic/ <b>revised</b>	
revisionid	topic/ph bookmap/ <b>revisionid</b>	Text Map Inline
row	topic/ <b>row</b>	TabRow
screen	topic/pre ui-d/ <b>screen</b>	Text Pre
searchtitle	map/ <b>searchtitle</b>	Text Map
searchtitle	topic/ <b>searchtitle</b>	Text
section	topic/ <b>section</b>	Text Section
sectiondiv	topic/ <b>sectiondiv</b>	Text Section
sep	topic/ph pr-d/ <b>sep</b>	Text
series	topic/ <b>series</b>	Text Var
shape	topic/keyword ut-d/ <b>shape</b>	Text Image
shortcut	topic/keyword ui-d/ <b>shortcut</b>	Text Inline
shortdesc	map/ <b>shortdesc</b>	Text SDesc Map
shortdesc	topic/ <b>shortdesc</b>	Text SDesc
simpletable	topic/ <b>simpletable</b>	Table TabStart
sl	topic/ <b>sl</b>	List
sli	topic/ <b>sli</b>	Text ListItem
source	topic/ <b>source</b>	Text Var
started	topic/ph bookmap/ <b>started</b>	Text Map Inline
state	topic/ <b>state</b>	Inline
stentry	topic/ <b>stentry</b>	Text TabCell
step	topic/li task/ <b>step</b>	Task Text ListItem XRSOURCE Pernicious
stepresult	topic/itemgroup task/ <b>stepresult</b>	Task Text Inline
steps	topic/ol task/ <b>steps</b>	Task List Num
steps-informal	<b>topic/section task/steps-informal</b>	Text Section Task
steps-unordered	topic/ul task/ <b>steps-unordered</b>	Task List
stepsection	topic/li task/stepsection	Task Text ListItem NoNumber Pernicious
stepxmp	topic/itemgroup task/ <b>stepxmp</b>	Task Text Inline
sthead	topic/ <b>sthead</b>	TabRow TabHead
strow	topic/ <b>strow</b>	TabRow TabBody
sub	topic/ph hi-d/ <b>sub</b>	Text Inline Typo
substep	topic/li task/ <b>substep</b>	Task Text ListItem XRSOURCE Pernicious
substeps	topic/ol task/ <b>substeps</b>	Task List Num Sub
summary	topic/ph bookmap/ <b>summary</b>	Text Map Inline

**Table B-1 Default properties assigned to elements (continued)**

Element name	Class attribute	Default element type properties
sup	topic/ph hi-d/ <b>sup</b>	Text Inline Typo
synblk	topic/figgroup pr-d/ <b>synblk</b>	
synnote	topic/fn pr-d/ <b>synnote</b>	Text
synnoteref	topic/xref pr-d/ <b>synnoteref</b>	
synph	topic/ph pr-d/ <b>synph</b>	Text Inline
syntaxdiagram	topic/fig pr-d/ <b>syntaxdiagram</b>	Inline
systemoutput	topic/ph sw-d/ <b>systemoutput</b>	Text Inline
table	topic/ <b>table</b>	Table TabStart
tablelist	map/topicref bookmap/ <b>tablelist</b>	Map Ref Topic List
task	topic/topic task/ <b>task</b>	Root Task
taskbody	topic/body task/ <b>taskbody</b>	
tbody	topic/ <b>tbody</b>	Table TabBody
term	topic/ <b>term</b>	Text Inline Glossary
tested	topic/data bookmap/ <b>tested</b>	Map
text	topic/text	Text Inline
tgroup	topic/ <b>tgroup</b>	Table
thead	topic/ <b>thead</b>	Table TabHead
title	topic/ <b>title</b>	Text Title XRSOURCE
titlealts	topic/ <b>titlealts</b>	
tm	topic/ <b>tm</b>	Text Inline
toc	map/topicref bookmap/ <b>toc</b>	Map Ref Topic List
topic	topic/ <b>topic</b>	Root
topicgroup	map/topicref mapgroup-d/ <b>topicgroup</b>	Map Group Topic
topichead	map/topicref mapgroup-d/ <b>topichead</b>	Map Topic
topicmeta	map/ <b>topicmeta</b>	Map Meta Topic
topicref	map/ <b>topicref</b>	Map Ref Topic
topicset	map/topicref mapgroup-d/ <b>topicset</b>	Map Ref Topic
topicsetref	map/topicref mapgroup-d/ <b>topicsetref</b>	Map Ref Topic
trademarklist	map/topicref bookmap/ <b>trademarklist</b>	Map Ref Topic List
tt	topic/ph hi-d/ <b>tt</b>	Text Inline
tutorialinfo	topic/itemgroup task/ <b>tutorialinfo</b>	Task Text Inline
u	topic/ph hi-d/ <b>u</b>	Text Inline Typo
uicontrol	topic/ph ui-d/ <b>uicontrol</b>	Text Inline CascadeItem
ul	topic/ <b>ul</b>	List
unknown	topic/ <b>unknown</b>	Inline
userinput	topic/ph sw-d/ <b>userinput</b>	Text Inline
var	topic/ph pr-d/ <b>var</b>	Text
varname	topic/keyword sw-d/ <b>varname</b>	Text Inline
volume	topic/data bookmap/ <b>volume</b>	Map
vrml	topic/ <b>vrml</b>	
vrmlist	topic/ <b>vrmlist</b>	

**Table B-1 Default properties assigned to elements (continued)**

Element name	Class attribute	Default element type properties
wintitle	topic/keyword ui-d/ <b>wintitle</b>	Text Inline
xref	topic/ <b>xref</b>	Text Inline Ref
year	topic/ph bookmap/ <b>year</b>	Text Map Inline

# C 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:*

§41 [Working with content models](#) on page 753

```
; ContentModel.txt describes sections used in DITASpecial.ini files,
; such as DITAconcept11.ini, as they are supported in dwhtm.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 mif2htm.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=nameest
; 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

## 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

AddCntFileName, [HelpContents] keyword 309

AddContentsLink, [RelatedLinks] keyword 191

AddedDividerFormat, [RelatedLinks]  
keyword 195

AddedDividerText, [RelatedLinksText]  
keyword 195

AddedLinksEnd, [RelatedLinks] keyword 195

AddedLinksFormat, [RelatedLinks]  
keyword 195

AddedLinksSpacer, [RelatedLinks]  
keyword 195

AddedLinksStart, [RelatedLinks] keyword 195

AddGlossaryLink, [RelatedLinks] keyword 191

AddIndexLink, [RelatedLinks] keyword 191

AddLOFLink, [RelatedLinks] keyword 191

AddLOTLink, [RelatedLinks] keyword 191

AKey, [HelpStyles] format property 260, 296, 304

ALink, [MarkerTypes] property 723

Altura, [HelpOptions] keyword 283

AncestorHead, [RelatedLinksText]  
keyword 194

[AnumCodeAfter], code after paragraph autonumber  
placement properties 712  
subject to configuration overrides 772

[AnumCodeBefore], code before paragraph auto-  
number  
placement properties 712  
subject to configuration overrides 772

AnumTabWidth, [WordOptions] keyword 226

AppendFlagsFile, ConditionOptions  
keyword 166

AppendixStream, [NumberStreams] keyword 147

AppendLinksToTopics, [RelatedLinks]  
keyword 191

Archive\*, [Automation] keywords:  
ArchiveCommand 804  
activated by WrapAndShip 788  
ArchiveEndParams 804  
ArchiveExt 805

ArchiveName 805

ArchiveStartParams 804

ArchiveVer 805

[AttributePrefixes], format-name prefixes  
based on attributes 96

AutoBrowse, [HelpBrowse] keyword 310

[Automation]

default values in local\_omsys.ini 33

default values supplied by Project Manager 43

produce deliverables 788

system commands 777, 778

### B

BaseMapFormat, [ElementOptions] keyword 105

[BaseValues], format property units 120

[BctFileHeads], WinHelp section 307

[\*BLForms], booklist item levels 217

[\*BLList], variant booklist identifiers 215

[\*BLItems], booklist item properties 216

[BlockFormatMaps], map block element paths to  
formats 93

[BlockFormatPrefixRunins], specify run-in  
headings 97

[BlockOutclassMaps], assign format names to  
block elements 90

[\*BLRefForms], formats for variant indexlist  
references 217

[\*BLText], title text for variant booklists 216

BodyBaseSize, [BaseValues] keyword 120

BodyBaseWidth, [BaseValues] keyword 120

[BookLists], name booklist components 214

BookmarkIXRanges, [WordOptions] keyword 221

Bottom, [Inserts] Word keyword 711

Browse, [HelpStyles] format property 296

[BrowsePrefix], WinHelp section 311

[BrowseStart], WinHelp section 311

Build, [HelpStyles] format property 296

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

## C

- CascadeSeparator, [ElementText] keyword 159
- CaselessMatch, [Options] keyword 73
- CatalogKeys, [Catalogs] keyword 67
- [Catalogs], XML catalog keys 67
- ChapterStream, [NumberStreams] keyword 147
- [Char], default character format 122
- [CharStyle\*] sections
  - [CharStyleCode\*] sections
    - all subject to configuration overrides* 772
    - [CharStyleCodeAfter] 712
    - [CharStyleCodeBefore] 712
    - [CharStyleCodeEnd] 712
    - [CharStyleCodeReplace] 712
    - [CharStyleCodeStart] 712
- CharStylesUsedInText
  - [WordOptions] keyword 227
- CharTags, [WordOptions] keyword 227
- ChildHead, [RelatedLinksText] keyword 194
- ChoiceTableFormat, [TableOptions] keyword 103
- Cnt\*, [HelpContents] keywords:
  - CntBase 307
  - CntBStyleText 308, 309
  - CntMainWindow 309
  - CntName 307
  - CntStartFile 307
  - CntTitle 307
  - CntTopHead 307
  - CntTopic 307
  - CntType 249, 306
- Code\*, [HelpStyles] and [WordStyles] format properties
  - CodeAfter 693, 712
  - CodeAfterAnum 712
  - CodeBefore 693, 712
  - CodeBeforeAnum 712
  - CodeEnd 712
  - CodeReplace 712
  - CodeStart 693, 712
  - CodeStore 688, 693
- Code, [MarkerTypes] property 723
- CodePage, [Defaults] keyword 220
- CompactForm. [\*BList] keyword 216
- CompileHelp, [Automation] keyword 286
  - compile WinHelp project 802
  - set up WinHelp project 284
- Compiler, [HelpOptions] keyword 285
- ComplexOtherprops, ConditionOptions keyword 162
- ConceptsHead, [RelatedLinksText] keyword 194
- [CondEndFlagAltText], specify flag alt text 168
- [CondEndFlagImages], specify flag images 168
- ConditionalDefaults, ConditionOptions keyword 163
- [ConditionalExclude], exclude content 164
- [ConditionalFlagging], flag content 163
- [ConditionalFlags], specify flag properties 166
- [ConditionalPassthrough], pass attributes through 165
- [ConditionAttributes], assign attributes with flags 169
- [ConditionOptions], process otherprops 162
- [ConditionOptions], set flags 165
- [ConditionOptions], specify ditaval file 161
- [CondStartFlagAltText], specify flag alt text 168
- [CondStartFlagImages], specify flag images 168
- Config, override configuration settings
  - [HelpStyles] format property 776
  - [MarkerTypes] property 723
  - [WordStyles] format property 776
- Configs, [Templates] keyword 731, 737, 740, 742, 816
  - chain of templates 743
  - precedence of settings 765
- Contents, [HelpStyles] format property 250, 296
- [Contents]
  - generate a TOC 221
- [ContentsText]
  - title and other fixed text 199
- [ContentsText], contents entries and links 191
- ContinuedFormatSuffix, [ElementOptions] keyword 89
- CopyAfterFiles, [Automation] keyword 795
- CopyAfterFrom, [Automation] keyword 795
- CopyBeforeFiles, [Automation] keyword 791
- CopyBeforeFrom, [Automation] keyword 791
- CopyGraphicsFrom, [Automation] keyword 797

activated by CompileHelp [803](#)  
 activated by WrapAndShip [788](#)

CopyOriginalGraphics, [Automation]  
 keyword [796](#)  
 CousinHead, [RelatedLinksText] keyword [194](#)  
 CSSFlagsFile, ConditionOptions keyword [165](#)

## D

DefaultBlockFormat, [ElementOptions]  
 keyword [88](#)  
 DefaultInlineFormat, [ElementOptions]  
 keyword [88](#)  
 DefaultNoteType, [ElementOptions]  
 keyword [101](#)  
 [Defaults] [220](#), [227](#)  
 subject to configuration overrides [770](#)  
 DefinitionListTables, [TableOptions]  
 keyword [104](#)  
 DefListTableColWidths, [TableOptions]  
 keyword [104](#)  
 DefListTableFormat, [TableOptions]  
 keyword [104](#)  
 Delete  
 [HelpStyles] format property [296](#)  
 [MarkerTypes] property [723](#)  
 [XrefStyles] format property [290](#)  
 Delete, [WordStyles] format property [228](#)  
 DeleteExistingDCL, [Setup] keyword [73](#)  
 DescendantHead, [RelatedLinksText]  
 keyword [194](#)  
 Digits, [HelpBrowse] keyword [310](#)  
 DisambiguateIndex, [HelpOptions]  
 keyword [306](#)  
 DisplayElementPath, [ElementOptions]  
 keyword [73](#)  
 DitavalFile, ConditionOptions keyword [161](#)  
 Document, [Templates] keyword [732](#), [737](#), [738](#)  
 [Document], output page properties [134](#)  
 DTDPPath, [Setup] keyword [33](#), [69](#)  
 DuplicateNameCheck, [IDOptions] keyword [77](#)

## E

EditorFileName, [Logging] keyword [74](#)  
 [ElementAttrPrefixes], prefix format names  
 based on attributes [95](#)  
 [ElementClasses], add ^class to elements [183](#)  
 [ElementOptions], assign formats to elements [88](#)  
 ElementPathFormat, [ElementOptions]  
 keyword [73](#)  
 [ElementSets], define sets of elements [179](#)  
 [ElementText], miscellaneous text  
 assignments [159](#)  
 [ElementTypes], properties of element types [183](#)  
 EmptyFrames  
 [HelpOptions] keyword [751](#)  
 [WordOptions] keyword [751](#)  
 EmptyGraphPath, [Automation] keyword  
 activated by CompileHelp [803](#)  
 EmptyOutputDir, [Automation] keyword [789](#)  
 activated by CompileHelp [803](#)  
 dependencies [791](#)  
 when effective [790](#)  
 EmptyOutputFiles, [Automation] keyword [790](#)  
 activated by CompileHelp [803](#)  
 when to include [791](#)  
 EmptyWrapPath, [Automation] keyword [793](#)  
 activated by CompileHelp [803](#)  
 dependencies [799](#)  
 [End], dummy section to end settings [66](#)  
 [End], dummy section to replace  
 [MacroVariables] [680](#)  
 EndFtnWithSpace, [HelpOptions] keyword [284](#)  
 ExactLineSpace, [WordOptions] keyword [227](#)  
 ExtendHelpNoScroll, [HelpOptions]  
 keyword [297](#), [298](#)  
 ExternalXrefs, [WordOptions] keyword [221](#)

## F

FieldHyper, [WordOptions] keyword  
*(deprecated)* [230](#)  
 FigTitleAboveImage, [FigureOptions]  
 keyword [102](#)  
 FigTitleAboveTable, [FigureOptions]  
 keyword [102](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

FigureAnchorFormat, [FigureOptions]  
 keyword [103](#)

[FigureOptions], title placement [102](#)

File, [HelpStyles] format property [296](#)

[FileIDs], RTF bookmark prefix [77](#)

FileName, [\*BList] keyword [215](#)

FileNames, [Graphics] keyword [749](#)  
 reference WinHelp hypergraphics [300](#)  
 replace file extensions [78](#)  
 substitute files [235](#), [749](#)  
 synchronize settings [800](#)

FilePaths, [Graphics] keyword [749](#)  
 for already converted files [751](#)  
 for referenced graphics [235](#)  
 reference WinHelp hypergraphics [300](#)  
 substitute files [749](#), [750](#)  
 synchronize settings [800](#)

FileSuffix, [Setup] keyword [219](#)

FirstFooter, [Inserts] Word keyword [711](#)

FirstHeader, [Inserts] Word keyword [711](#)

Footer, [Inserts] Word keyword [711](#)

FootnoteFormat, [footnotes] keyword [102](#)

Footnotes  
 [HelpOptions] keyword [288](#)  
 [Footnotes], assign footnote formats [102](#)

FootnoteSeparator  
 [HelpOptions] keyword [288](#)

FootnoteSpace, [HelpOptions] keyword [284](#)

FootnoteStream, [NumberStreams] keyword [147](#)

ForceBmc, [HelpOptions] keyword [283](#)

ForceTableLineBreaks, [Tables] keyword [291](#)

[FormatAliases]  
 map format names [52](#), [113](#)  
 map subformat names [142](#)

Formats, [Templates] keyword [110](#), [120](#), [730](#)

FrameEndPara, [WordOptions] keyword [228](#)

FriendHead, [RelatedLinksText] keyword [194](#)

FullIndexRanges, [Index] keyword [209](#)

## G

GenerateALinks, [RelatedLinks] keyword [192](#)

GenerateIDX, [Index] keyword [222](#)

GenerateMapIfMissing, [MapGeneration]

keyword [69](#)

GenerateParentChild, [RelatedLinks]  
 keyword [190](#)

GeneratePrevNext, [RelatedLinks]  
 keyword [190](#)

GenerateSiblings, [RelatedLinks]  
 keyword [190](#)

GenerateTOC, [Contents] keyword [221](#)

GenerateUIDs, [IDOptions] keyword [76](#)

GenListXrefFormat, [ElementOptions]  
 keyword [102](#)

[Glossary], define glossary output [204](#)

[GlossaryText], text for title of generated  
 glossary [204](#)

GLSTitle, [GlossaryText] keyword [204](#)

GLSTitleFormat, [Glossary] keyword [204](#)

GLSTOCFormat, [Glossary] keyword [204](#)

GraphCopyFiles, [Automation] keyword [797](#)  
 activated by CompileHelp or FTSCCommand [803](#)

[GraphFiles], replace graphics files [78](#), [235](#), [749](#)  
 reference WinHelp hypergraphics [300](#)  
 synchronize graphics settings [800](#)

GraphicAlignment, [FigureOptions]  
 keyword [103](#)

[Graphics]  
 omit [751](#)  
 subject to configuration overrides [770](#)

Green, [HelpStyles] format property [296](#)

## H

HeadBlockFormat, [RelatedLinks] keyword [194](#)

Header, [Inserts] Word keyword [711](#)

HeadInlineFormat, [RelatedLinks]  
 keyword [194](#)

[HelpBrowse] [310](#), [311](#)  
 subject to configuration overrides [770](#)

[HelpContents] [306–310](#)  
 subject to configuration overrides [770](#)

HelpCopyDate, [HelpOptions] keyword [286](#)

HelpCopyright, [HelpOptions] keyword [286](#)

HelpLineBreak, [HelpOptions] keyword [298](#)

[HelpOptions]  
 cross references [288](#), [289](#), [290](#)



- footnotes [284, 288](#)
- graphics [751](#)
- index [250, 253, 305, 306](#)
- links [301](#)
- options determined at run time, *listed* [733](#)
- page and section breaks [283, 285](#)
- platforms [283](#)
- subject to configuration overrides [770](#)
- tables [290, 291, 292](#)
- titles [298](#)
- [HelpReplacements] [287](#)
  - subject to configuration overrides [772](#)
- HelpSectionBreaks, [HelpOptions]
  - keyword [285](#)
  - platform differences [283](#)
- [HelpStyles]
  - “A” footnotes [260](#)
  - ALinks and keywords [304](#)
  - basic properties [284](#)
  - replace content [287](#)
  - subject to configuration overrides [772](#)
- [Help\*Styles], WinHelp sections
  - [HelpBrowsePrefixStyles] [311](#)
  - [HelpCntStyles]
    - basic conversion options [284](#)
    - understand level numbers [250, 308](#)
  - [HelpKeywordStyles] [305](#)
  - [HelpMacroStyles] [296, 303](#)
  - [HelpSuffixStyles] [297](#)
  - [HelpTitleSufStyles] [297, 298](#)
  - [HelpTopicBuildStyles] [296](#)
  - [HelpWindowStyles] [263, 297, 302](#)
- [HelpXrefFiles], cross references [289](#)
- Hide, [WordStyles] format property [228](#)
- HistoryFileName, [Logging] keyword [74](#)
- HPJFileName, [HelpOptions] keyword [284](#)
- HyperHelp, [HelpOptions] keyword [283](#)

## I

- IDElemSep, [IDOptions] keyword [77](#)
- IDFile, [IDOptions] keyword [76](#)
- [IDOptions] options for element IDs [76](#)
- IDPath, [IDOptions] keyword [76](#)
- IDPathSep, [IDOptions] keyword [76](#)
- IDTopic, [IDOptions] keyword [76](#)

- IDTopSep, [IDOptions] keyword [77](#)
- IDUpDir, [IDOptions] keyword [77](#)
- IdxColon, [HelpOptions] keyword [253, 305](#)
- IDXFormat, [Index] keyword [207](#)
- IDXTitle, [IndexText] keyword [191](#)
- IDXTitleFormat, [Index] keyword [207](#)
- IDXTitleFormat, [IndexText] keyword [207](#)
- IDXTOCFormat, [Index] keyword [207](#)
- [ImportancePrefixes], format-name prefixes
  - based on importance value [96](#)
- IncludeElementTags, [ElementOptions]
  - keyword [72](#)
- Index
  - [HelpOptions] keyword [250](#)
  - [WordOptions] keyword [240](#)
- [Index]
  - generate an index [222](#)
- [IndexClasses], map indexterms to indexlist
  - variants [218](#)
- IndexLetterNumber, [Index] keyword [210](#)
- IndexLettersFormat, [Index] keyword [210](#)
- IndexLetterSymbol, [Index] keyword [210](#)
- [IndexLists], assign indexterm outputclasses to indexlist
  - variants [218](#)
- IndexRangeSep, [IndexText] keyword [210](#)
- IndexRefSep, [IndexText] keyword [210](#)
- IndexRefStartSep, [IndexText] keyword [210](#)
- IndexSeeAlsoEnd, IndexSeeFormats]
  - keyword [208](#)
- IndexSeeAlsoStart, IndexSeeFormats]
  - keyword [208](#)
- IndexSeeEnd, IndexSeeFormats] keyword [208](#)
- [IndexSeeFormats], assign index see and see-also
  - formats [208](#)
- IndexSeeStart, IndexSeeFormats] keyword [208](#)
- [IndexSeeText], index see/see-also entries [209](#)
- IndexTopLettersFormat, [Index] keyword [211](#)
- [InlineFormatMaps], map inline element paths to
  - formats [93](#)
- [InlineFormatPrefixRunins], specify run-in
  - headings [98](#)
- [InlineOutclassMaps], assign format names to in-
  - line elements [90](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

[InlineOutclassMaps], for <xref> wrappers 91  
 [Inserts], insert code at predefined locations 711  
   subject to configuration overrides 771  
 ItemFormat. [\*BList] keyword 215

## J

JumpHot, [HelpStyles] format property 296  
 JumpTarget, [HelpStyles] format property 295

## K

KeepDraftComments, [ElementOptions]  
   keyword 105  
 KeepSectBreaks  
   [HelpOptions] keyword 285  
 Key, [HelpStyles] format property 296, 304  
 KeydefsOnlyWithinBranch, KeyOptions  
   keyword 172  
 [KeyOptions], limit scope of keydefs 172  
 KeywordLimit, [HelpOptions] keyword 252

## L

Language, [Defaults] keyword 220  
 Languages, [Templates] keyword 158, 730  
 LeftFooter, [Inserts] Word keyword 711  
 LeftHeader, [Inserts] Word keyword 711  
 LineSpacing, RTF [Defaults] keyword 227  
 LinkFormat, [ElementOptions] keyword 89  
 [ListOfFiguresText], text for generated LOF 201  
 [ListOfTables], define LOT output 202  
 [ListOfTablesText], text for generated LOT 202  
 [ListOptions], list styles 127  
 ListStream, [NumberStreams] keyword 147  
 ListTitle. [\*BLText] keyword 216  
 Local, [HelpStyles] format property 296  
 LockAllNavtitles, [MapOptions] keyword 200  
 LockHyper, [WordOptions] keyword 231, 239  
 LockXrefs, [WordOptions] keyword 229, 239  
 LOFFormat, [ListOfFigures] keyword 202  
 LOFTitle, [ListOfFiguresText] keyword 201

LOFTitleFormat, [ListOfFigures] keyword 201  
 LOFTOCFormat, [ListOfFigures] keyword 202  
 LOFXrefFormat, [ListOfFigures] keyword 202  
 LogAuto, [Automation] keyword 788  
 LogDebug, [Logging] keyword 74  
 LogErrors, [Logging] keyword 74  
 LogFileName, [Logging] keyword 74  
 [Logging] conversion events 74  
 LogInfo, [Logging] keyword 74  
 LogIniChains, [Logging] keyword 75  
 LogQuerys, [Logging] keyword 74  
 LogWarnings, [Logging] keyword 74  
 LOTFormat, [ListOfTables] keyword 203  
 LOTTitle, [ListOfTablesText] keyword 202  
 LOTTitleFormat, [ListOfTables] keyword 202  
 LOTTOCFormat, [ListOfTables] keyword 203  
 LOTXrefFormat, [ListOfTables] keyword 203

## M

Macro, [HelpStyles] format property 296  
 MacroHot, [HelpStyles] format property 303  
 MacroNestMax, [Macros] keyword 683, 705  
 Macros, [Templates] keyword 684, 729  
 [Macros]  
   debug 709  
   loop-control limits 705  
   remove implicit line breaks 681  
 [MacroTemplates], assign a template macro to a  
   file 187  
 [MacroVariables] 689  
   create a macro variable 688  
 MacroVarNesting, [Macros] keyword 690  
 MakeArchive, [Automation] keyword 43  
 MakeCombinedCnt, [HelpOptions] keyword 284  
   determined at run time 733  
 MakeRef, [HelpStyles] format property 296  
   for pop-up graphics 293  
 [MapGeneration], produce DITA map from  
   topics 69  
 MapLanguage, [MapGeneration] keyword 70  
 [MapOptions], configure generated maps 106, 200  
 MapRootElem, [MapGeneration] keyword 69

MapTitle, [MapGeneration] keyword 69  
 MapTitleElem, [MapGeneration] keyword 69  
 [MarkerTypeCodeAfter] 725  
 [MarkerTypeCodeBefore] 724  
 [MarkerTypeCodeReplace] 725  
 [MarkerTypes], marker-type properties 723  
 MergeStradCells  
   [Table] keyword 234  
   [Tables] keyword 292  
 MoveArchive, [Automation] keyword 807

## N

NameGraphics, [Graphics] keyword 237  
 NameUndefinedMacros, [Macros] keyword 709  
 NameUndefinedMacroVars, [Macros]  
   keyword 709  
 NextHead, [RelatedLinksText] keyword 194  
 NoMemDel, [Options] keyword 821  
 NoNameDel, [Options] keyword 821  
 NormalTableFormat, [TableOptions]  
   keyword 103  
 NoScroll, [HelpStyles] format property 296  
 NoSeeAlso  
   [HelpOptions] keyword 306  
   [WordOptions] keyword 240  
 [NoteAttrPrefixes], attributes to determine which  
   Note format to use 96  
 NoTitle, [HelpStyles] format property 296  
   for pop-up topics 295  
 NoXrefPopups, [HelpOptions] keyword 301  
 NoXScroll, [HelpStyles] format property 297  
   for pop-up topics 295  
 [NumberFormatsText], text for numbering  
   formats 149

## O

OmitMacroReturns, [Macros] keyword 681  
 OnlyAuto, [Automation] keyword 808  
 [Options]  
   debug 821  
   for cases, spaces, and wildcards 73  
   for conversion debugging 821

subject to configuration overrides 771  
 OutputclassHasBorderShadeFormats,  
   [ElementOptions] keyword 88  
 OutputclassHasBorderShadeFormatst,  
   [ElementOptions] keyword 145

## P

PageBreaks  
   [HelpOptions] keyword 285  
 Pages, [Setup] keyword 134  
 Pages, [Templates] keyword 110, 730  
 ParaLink, [HelpStyles] format property 296  
 ParameterListTables, [TableOptions]  
   keyword 104  
 ParamListTableColWidths, [TableOptions]  
   keyword 104  
 ParamListTableFormat, [TableOptions]  
   keyword 104  
 [Parar], default paragraph format 122  
 [ParaStyle\*] sections  
   [ParaStyleCode\*] sections  
     *all subject to configuration overrides 772*  
     [ParaStyleCodeAfter] 712  
     [ParaStyleCodeBefore] 712  
     [ParaStyleCodeEnd] 712  
     [ParaStyleCodeReplace] 712  
     [ParaStyleCodeStart] 712  
 ParentHead, [RelatedLinksText] keyword 194  
 PartStream, [NumberStreams] keyword 147  
 [PeerLinks], resolve peer related links 196  
 PicScale[WordOptions] keyword 236  
 Pop\*, [HelpStyles] format properties  
   PopContent 295  
   PopHot 296  
   PopOver 296, 297, 300  
 Prefix, [HelpBrowse] keyword 311  
 PrevHead, [RelatedLinksText] keyword 194  
 PrevRef, [HelpStyles] format property 297  
   for pop-up graphics 294  
 PrintProject, [Setup] keyword 71  
 PropertiesTableFormat, [TableOptions]  
   keyword 103  
 PublicID, [MapGeneration] keyword 70

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

## Q

### Quotes

ElementText] keyword 143, 159  
[WordOptions] keyword 227

## R

Refer, [HelpStyles] format property 297  
ReferenceFlagsFile, ConditionOptions  
keyword 165  
ReferencesHead, [RelatedLinksText]  
keyword 194  
RefFormat. [\*BList] keyword 216  
RefID. [\*BList] keyword 215  
RelatedDividerFormat, [RelatedLinks]  
keyword 195  
RelatedHead, [RelatedLinksText] keyword 194  
[RelatedLinks], append links to topics 191  
[RelatedLinksText], labels for related links 194  
RemoveGraphics, [Graphics] keyword 751  
RepeatMax, [Macros] keyword 706  
Replace  
[HelpStyles] format property 287, 297  
[WordStyles] format property 228  
Replace, [HelpStyles] format property 296  
[Required], include unused formats 120  
ResetAbbrevAt, [Glossary] keyword 205  
Resume, [HelpStyles] format property 293, 295,  
296, 297  
RevProt, [WordOptions] keyword 239  
RevTrack, [WordOptions] keyword 239  
RightFooter, [Inserts] Word keyword 711  
RightHeader, [Inserts] Word keyword 711  
RTFConfig  
[HelpStyles] format property 776  
[MarkerTypes] property 723  
[WordStyles] format property 776  
[RuninHeadText], text of run-in headings 98, 154

## S

Scope, [Templates] keyword 734, 741  
Scroll, [HelpStyles] format property 297, 298

for pop-up topics 295  
SeeAlsoEndIndex, [IndexSeeText] keyword 209  
SeeAlsoStartIndex[IndexSeeText]  
keyword 209  
SeeEndIndex, [IndexSeeText] keyword 209  
SeeStartIndex, [IndexSeeText] keyword 209  
SeqAnums, [WordOptions] keyword 226  
[Setup]  
compile WinHelp 284  
options determined at run time, *listed* 733  
subject to configuration overrides 770  
[Setup], set up DITA2Go options 33, 69  
ShiftWideTablesLeft  
[Tables] keyword 290  
ShipPath, [Automation] keyword 806  
activated by WrapAndShip 788  
ShortdescFormat, [RelatedLinks] keyword 194  
ShowElementPath, [ElementOptions]  
keyword 72  
ShowLog, [Logging] keyword 74  
SiblingHead, [RelatedLinksText] keyword 194  
SimpleTableFormat, [TableOptions]  
keyword 103  
Slide, [HelpStyles] format property 294, 297  
SlideEnd, [Inserts] WinHelp keyword 711  
SlideStart, [Inserts] WinHelp keyword 711  
SpaceAfterUnicode, [Defaults] keyword 221  
SpacelessMatch, [Options] keyword 63, 73  
SpKey, [HelpStyles] format property 296, 297,  
304  
Start, [HelpBrowse] keyword 311  
Step, [HelpBrowse] keyword 310  
[StepAttrPrefixes], attributes to determine which  
Step format to use 96  
[StepImportancePrefixes], Step format-name  
prefixes based on importance value 96  
StepsHeadFormat, [ElementOptions]  
keyword 88  
StrippedCellPar, [Table] keyword 292  
StripTables, [Table] keyword 292  
StripTables, [Tables] keyword 294  
[StyleCodeStore], assign macro variable to para-  
graph format 693

[StyleReplacements], merge formats [226](#)  
 [Styles], map paragraph formats to Word styles [225](#)  
 SubFormats, [Templates] keyword [110](#), [141](#)  
 Subformats, [Templates] keyword [730](#)  
 Suffix, [HelpStyles] format property [297](#)  
 SystemCommandWindow, [Automation] keyword [778](#)  
 SystemEndCommand, [Automation] keyword [777](#)  
 SystemID, [MapGeneration] keyword [70](#)  
 SystemStartCommand, [Automation] keyword [777](#)  
 SystemWrapCommand, [Automation] keyword [777](#)

## T

TableAnchorFormat, [TableOptions] keyword [103](#)  
 [TableAnchorFormats], override anchor paragraph format for selected tables [103](#)  
 TableFill  
   [Table] keyword [233](#)  
   [Tables] keyword [291](#)  
 TableFooterClass, [TableOptions] keyword [104](#)  
 TableGraphics  
   [Tables] keyword [291](#)  
 [TableOptions], assign format names to tables [103](#)  
 [TableOptions], output options for tables [104](#)  
 [TableOutclassMaps], assign format names to tables [90](#)  
 TableRules  
   [Table] keyword [233](#)  
   [Tables] keyword [291](#)  
 Tables, [Templates] keyword [110](#), [129](#), [730](#)  
 TableTitles  
   [Table] keyword [233](#), [291](#)  
 TableWidthsFixed, [Table] keyword [290](#)  
 TasksHead, [RelatedLinksText] keyword [194](#)  
 TblColWidAdd, [Table] keyword [292](#)  
 TblColWidPct, [Table] keyword [292](#)  
 TblFootformat, [Footnotes] keyword [102](#)  
 TblFootnoteStream, [NumberStreams] keyword [147](#)

TblFullWidth, [Table] keyword [292](#)  
 Template, [WordOptions] keyword [223](#)  
 [Templates] [737](#)  
   for document-specific settings [732](#)  
   for general configuration settings [731](#)  
   for macro libraries [729](#)  
 TitleFormat. [\*BList] keyword [215](#)  
 TitleIndent, [HelpOptions] keyword [298](#)  
 TitleOnlyTopicID, [TopicHeads] keyword [106](#)  
 TitleOnlyTopicType, [TopicHeads] keyword [106](#)  
 TitleScroll, [HelpOptions] keyword [298](#)  
 TitleSpace, [HelpOptions] keyword [298](#)  
 TitleSuf, [HelpStyles] format property [297](#)  
 TOCFormat. [\*BList] keyword [216](#)  
 TOCFormat, [Contents] keyword [200](#)  
 TOCTitle, [ContentsText] keyword [191](#), [199](#)  
 TOCTitleFormat, [Contents] keyword [200](#)  
 TOCXrefFormat, [Contents] keyword [200](#)  
 Top, [Inserts] Word keyword [711](#)  
 Topic, [HelpStyles] format property [295](#), [297](#)  
   for pop-up topics [295](#)  
 TopicEnd, [Inserts] WinHelp keyword [711](#)  
 [TopicHeads] [200](#)  
 [TopicHeads], configure map output [106](#)  
 TopicheadsHaveNavtitles, [TopicHeads] keyword [200](#)  
 [TopicHeadText], heading for list of child topics [107](#)  
 TopicStart, [Inserts] WinHelp keyword [711](#)  
 TopicTitleFormat, [RelatedLinks] keyword [194](#)  
 TreatTableFigAsTable, [FigureOptions] keyword [203](#)

## U

Uline, [HelpStyles] format property [297](#)  
 UniqueNameSuffixFormat, [IDOptions] keyword [77](#)  
 UniqueNameSuffixLength, [IDOptions] keyword [77](#)  
 UseAbbrevInTitles, [Glossary] keyword [206](#)



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

UseAddedDivider, [RelatedLinks] keyword 195

UseAllInTOC, [MapOptions] keyword 201

UseAncestors, [RelatedLinks] keyword 190

UseBranchKeydefs, KeyOptions keyword 172

UseCatalogs, [Catalogs] keyword 67

UseChildren, [RelatedLinks] keyword 190

UseCompactForm, [Index] keyword 209

UseConditionalFlagging, ConditionOptions keyword 165

UseCousins, [RelatedLinks] keyword 190

UseDCLOutput, [Automation] keyword 44

UseDCLSource, [Automation] keyword 43

UseDescendants, [RelatedLinks] keyword 190

UseElementNameForFormat, [ElementOptions] keyword 88

UseExistingDCL, [Setup] keyword 73

UseFigureAnchor, [FigureOptions] keyword 103

UseFriends, [RelatedLinks] keyword 190

UseFullPath, [Options] keyword 70

UseGreen, [HelpOptions] keyword 300

UseHeading, [\*BList] keyword 215

UseHyperlinks  
[HelpOptions] keyword 301  
[WordOptions] keyword 231

UseIndexLeader, [Index] keyword 210

UseIndexLetters, [Index] keyword 210

UseLeader, [\*BList] keyword 216

UseLetters, [\*BList] keyword 216

UseLog, [Logging] keyword 74

UseMapDescAsTitle, [MapOptions] keyword 106

UseNestedTopicsInTOC, [Contents] keyword 199

UseOutputClassForFormat, [ElementOptions] keyword 88, 145

UseParent, [RelatedLinks] keyword 190

UsePrevNext, [RelatedLinks] keyword 190

UseRelatedDivider, [RelatedLinks] keyword 195

UseRelDescAsTitle, [RelatedLinks] keyword 192

UseRelDescription, [RelatedLinks] keyword 191

UseSiblings, [RelatedLinks] keyword 190

UseTableAnchor, [TableOptions] keyword 103

UseTOCDescriptions, [Contents] keyword 200

UseTopicShortdesc, [MapGeneration] keyword 69

UseTopicTypes, [RelatedLinks] keyword 191

UseTopLetters, [\*BList] keyword 216

## V

[VariableMaps], assign variable names to elements 186

ViewOutputCommand, [\*Options] keyword 35, 223, 248, 443

## W

WhileMax, [Macros] keyword 705

WildcardMatch, [Options] keyword 73

Window, [HelpStyles] format property 297, 302

Word2000, [WordOptions] keyword 224

Word2002, [WordOptions] keyword 224

Word2003, [WordOptions] keyword 224  
correct graphics scale 236

Word2007, [WordOptions] keyword 224

Word2009, [WordOptions] keyword 224

Word2010, [WordOptions] keyword 224

Word8, [WordOptions] keyword 224

[WordOptions]  
cross references 229, 232, 288  
formats 227  
graphics 751  
index 240  
line spacing 227  
special characters 227  
subject to configuration overrides 771  
tables 233

WordPerfect, [WordOptions] keyword 220

[WordReplacements] 228  
subject to configuration overrides 773

[WordStyles], print RTF format properties  
hide content 228  
omit content 228  
replace content 228  
subject to configuration overrides 773

WordSuffix, [Setup] keyword [223](#)  
[WordXrefFiles], cross references [232](#)  
WrapAndShip, [Automation] keyword [788](#)  
WrapCopyFiles, [Automation] keyword [793](#)  
    activated by CompileHelp [803](#)  
WrapPath, [Automation] keyword [792](#)  
    activated by CompileHelp [803](#)  
    activated by WrapAndShip [788](#)  
    for WinHelp [284](#), [286](#), [346](#)  
WrapXrefs, [WordOptions] keyword [230](#)  
WriteAnums  
    [HelpOptions] keyword [287](#)  
    [WordOptions] keyword [226](#)  
WriteFlagsFile, ConditionOptions  
    keyword [165](#)  
WriteHelpProjectFile, [HelpOptions]  
    keyword [283](#)

## X

XrefFileDefault, [HelpOptions] keyword [290](#)  
XrefFileSuffix  
    [HelpOptions] keyword [289](#)  
    [WordOptions] keyword [232](#)  
XrefFormat. [\*BList] keyword [216](#)  
XrefFtnFormat, [ElementOptions] keyword [101](#)  
XrefHyper, [WordOptions] keyword [229](#)  
XrefLenLimit, [HelpOptions] keyword [290](#)  
XrefNumFormat, [ElementOptions] keyword [101](#)  
[XrefOutclassMaps], format names for cross  
    references [91](#)  
Xrefs  
    [HelpOptions] keyword [288](#)  
    [WordOptions] keyword [229](#)  
[XrefStyles], cross-reference formats [231](#), [290](#)  
    subject to configuration overrides [773](#)  
XrefTextFormat, [ElementOptions]  
    keyword [101](#)  
XrefTitleFormat, [ElementOptions]  
    keyword [101](#)  
XrefType, [WordOptions] keyword [288](#)  
XScroll, [HelpStyles] format property [297](#)



ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

# 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 662
  - assign a value in [StyleCellAbbr] 663
  - cell content abbreviation 662
- AbbrVal, [HTMLParaStyles] WAI format
  - property 666
- AccessMethod, [Table] keyword 658
  - apply the id/headers method to all tables 659
  - apply the scope method to all tables 658
  - avoid redundant attribute assignments 447
  - effects on ColGroup property 676
  - effects on RowGroup property 677
  - overriding attributes 644
  - use the scope method to identify cells 667
  - WAI strategy for row/column markup 657
  - ways to override 659
- AddCntWindowName, [HelpContents]
  - keyword 309
- AddContentsLink, [RelatedLinks] keyword 191
- AddedDividerFormat, [RelatedLinks]
  - keyword 195
- AddedDividerText, [RelatedLinksText]
  - keyword 195
- AddedLinksEnd, [RelatedLinks] keyword 195
- AddedLinksFormat, [RelatedLinks]
  - keyword 195
- AddedLinksSpacer, [RelatedLinks]
  - keyword 195
- AddedLinksStart, [RelatedLinks] keyword 195
- AddGlossaryLink, [RelatedLinks] keyword 191
- AddIndexLink, [RelatedLinks] keyword 191
- AddLOFLink, [RelatedLinks] keyword 191
- AddLOTLink, [RelatedLinks] keyword 191
- address, [ParaTags] format property 566
- AddTopicHeadChildren, [TopicHeads]
  - keyword 107
- [AElement], implied link format 128
- AliasPrefix
  - [MSHtmlHelpOptions] keyword 343
  - [OmniHelpOptions] keyword 375
- AliasTitle, [MSHtmlHelpOptions] keyword 344
- AlignAttributes, [HTMLOptions] keyword
  - CSS-dependent default value 597
  - override paragraph properties 573
  - XML default value 449
- ALink
  - [HTMLParaStyles] format property
    - create HTML Help ALink buttons 329
    - for HTML-based Help 261
    - for Oracle Help 409
  - [MarkerTypes] property 723
- ALink\*, [MSHtmlHelpOptions] keywords:
  - ALinkButtonGraphic 326
  - ALinkButtonHeight 326
  - ALinkButtonIcon 326
  - ALinkButtonText 326
  - ALinkButtonWidth 326
  - ALinkEmptyTopic 326
  - ALinkFlags 326
  - ALinkText 326
  - ALinkTextFont 326
  - ALinkType 326
- ALinkRefs, [OmniHelpOptions] keyword 370
- ALinkRefs, [OracleHelpOptions] keyword 409
- AllowEmptyAlt, [Graphics] keyword 620
- AllowNobr, [HTMLOptions] keyword 447
- AllowTbSplit, [Table] keyword
  - convert tables to paragraphs 647
  - designate split points 526
- AllowTbTitle, [Table] keyword
  - convert tables to paragraphs 647
  - titles for split files 532
- Alt, [HTMLParaStyles] format property for alt attribute 651
- AlwaysNestLists, [CSS] keyword 587
- AncestorHead, [RelatedLinksText]
  - keyword 194
- ANSI, [MarkerTypes] property 723
- Anum, [HTMLParaStyles] format property, retain autonumbers 568
  - [AnumCodeAfter], code after paragraph autonumber indent list items 589
  - placement properties 712

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

subject to configuration overrides [772](#)

[AnumCodeBefore], code before paragraph auto-number

placement properties [712](#)

subject to configuration overrides [772](#)

AnumTabs, [HtmlOptions] keyword [568](#)

AppendFlagsFile, ConditionOptions  
keyword [166](#)

AppendixStream, [NumberStreams] keyword [147](#)

AppendLinksToTopics, [RelatedLinks]  
keyword [191](#)

Archive\*, [Automation] keywords:

ArchiveCommand

activated by WrapAndShip [788](#)

archive deliverables [804](#)

place deliverables [807](#)

ArchiveEndParams [804](#)

ArchiveExt [805](#)

ArchiveName [805](#)

ArchiveStartParams [804](#)

ArchiveVer [805](#)

ATagElement, [HTMLOptions] keyword [454](#)

ATagLineBreak, [HTMLOptions] keyword [439](#)

[AttributePrefixes], format-name prefixes  
based on attributes [96](#)

[Attributes]

for background images [624](#)

for <body> element [438](#)

for links [546](#)

for tables [632](#)

overridden by [TableAttributes] [633](#)

subject to configuration overrides [770](#)

when not to use [637](#)

[AUnvisitedElement], implied link format [129](#)

[Automation]

default values in local\_omsys.ini [33](#)

default values supplied by Project Manager [43](#)

pre- and post-conversion system code [777](#)

produce deliverables [788](#)

[AVisitedElement], implied link format [129](#)

Axis, [HTMLParaStyles] WAI format  
property [662](#)

AxisVal, [HTMLParaStyles] WAI format  
property [666](#)

## B

[Base], default font and size [577](#)

subject to configuration overrides [770](#)

Basefont, [HTMLOptions] keyword [577](#), [597](#), [609](#)

BaseMapFormat, [ElementOptions] keyword [105](#)

[BaseValues], format property units [120](#)

BeginFile, [Inserts] keyword [432](#)

BinaryIndex

[MSHtmlHelpOptions] keyword [335](#)

BinaryTOC

[MSHtmlHelpOptions] keyword [335](#)

[\*BLForms], booklist item levels [217](#)

[\*BList], variant booklist identifiers [215](#)

[\*BLItems], booklist item properties [216](#)

[BlockFormatMaps], map block element paths to  
formats [93](#)

[BlockFormatPrefixRunins], specify run-in  
headings [97](#)

[BlockOutclassMaps], assign format names to  
block elements [90](#)

Blockquote, [ParaTags] format property [566](#)

[\*BLRefForms], formats for variant indexlist  
references [217](#)

[\*BLText], title text for variant booklists [216](#)

BodyBaseSize, [BaseValues] keyword [120](#)

BodyBaseWidth, [BaseValues] keyword [120](#)

BodyContentOnly, [HTMLOptions] keyword [443](#)

[BodyElement], implied block format [120](#)

Bold, [HTMLParaStyles] or [HTMLCharStyles]  
format property [573](#)

[BookLists], name booklist components [214](#)

Border, [Table] keyword [636](#)

overridden by [TableAttributes] [633](#), [637](#)

Bottom, [Inserts] keyword [535](#), [536](#), [710](#)

position a navigation macro [564](#)

## C

CascadeSeparator, [ElementText] keyword [159](#)

CaselessMatch, [Options] keyword [73](#)

CaseSensitiveIndexCompare, [Index]  
keyword [257](#)

CatalogKeys, [Catalogs] keyword [67](#)

- [Catalogs], XML catalog keys [67](#)
- CellAlignAttributes, [Table] keyword [636](#)
- CellAttribute, [HTMLParaStyles] format property [635](#)
- CellAttribute, [HTMLParaStyles] format property [662](#)
- CellAttribute, [HTMLParaStyles] WAI format property [662](#)
- CellColorAttributes, [Table] keyword [636](#)
- Center, [HTMLParaStyles] format property [573](#)
- CGElems, [TableAccess] property [630](#)
- ChangeFileNameSpaces, [HTMLOptions] keyword [783](#)
- [Char], default character format [122](#)
- [CharacterRangeClasses], assign classes to Unicode character ranges [603](#)
- [CharClasses]
  - default use for CSS class names [592](#)
  - map character formats for XML [452](#)
  - map character formats to span classes [602](#)
  - subject to configuration overrides [770](#)
- [CharConvert], map special characters [574](#)
- [CharStyle\*] sections
  - [CharStyleCode\*] sections
    - all subject to configuration overrides* [772](#)
    - [CharStyleCodeAfter] [712](#)
    - [CharStyleCodeBefore] [712](#)
    - [CharStyleCodeEnd] [712](#)
    - [CharStyleCodeReplace] [712](#)
    - [CharStyleCodeStart] [712](#)
  - [CharStyleCSS] [608](#)
  - [CharStyleLinkSrc] [712](#)
- [CharTags]
  - assign HTML tags to character formats [569](#)
  - assign XML tags to character formats [452](#)
  - map character formats to CSS span classes [602](#)
  - subject to configuration overrides [772](#)
  - tags used for CSS classes by default [592](#)
- CheckAllRefs, [HTMLOptions] keyword [551](#)
- ChildHead, [RelatedLinksText] keyword [194](#)
- [ChmFiles], map source files to .chm files [324](#), [349](#)
- ChmFormat, [MSHtmlHelpOptions] keyword [324](#)
- ChoiceTableFormat, [TableOptions] keyword [103](#)
- ChunkBy, [Chunking] keyword [524](#)
- [Chunking] [523](#)
- ChunkSel, [Chunking] keyword [525](#)
- ClassIsTag, [CSS] keyword [601](#)
  - map CSS class names to XML tags [452](#)
  - use tag names for CSS class names [604](#)
  - XML default value [449](#)
- ClassSpaceChar, [HtmlOptions] keyword [600](#)
- ClickBlockToClose, [DropDowns] keyword [271](#)
- CloseOldWindow, [OmniHelpOptions] keyword [363](#)
- Code\*, [HTMLCharStyles] format properties
  - CodeAfter [712](#)
  - CodeBefore [712](#)
  - CodeEnd [712](#)
  - CodeReplace [712](#)
  - CodeStart [712](#)
- Code\*, [HTMLParaStyles] format properties
  - CodeAfter [693](#), [712](#)
  - CodeAfterAnum [712](#)
  - CodeBefore [693](#), [712](#)
  - CodeBeforeAnum [712](#)
  - CodeEnd [567](#), [693](#), [712](#)
  - CodeReplace [328](#), [712](#)
  - CodeStart [567](#), [693](#), [712](#)
  - CodeStore [693](#), [694](#)
    - capture FrameMaker autonumbers [780](#)
    - create a macro variable [688](#)
    - difference from TextStore [694](#)
- Code, [MarkerTypes] property [723](#)
- ColGroup, [HTMLParaStyles] WAI format property [662](#)
  - in [Table]ColGroupHead cells [670](#)
  - use header cells to define column groups [660](#)
- ColGroupElements, [Table] keyword [628](#)
  - apply scope method to all tables [658](#)
  - for browser-dependent table tags [628](#)
  - override column group settings [630](#)
- ColGroupHead, [Table] WAI keyword [670](#)
  - column-group extent [671](#)
  - id/header table cell attribute [670](#)
- ColGroupIDs, [Table] WAI keyword [670](#)
  - column-group extent [671](#)
  - id/header table cell attribute [670](#)
  - override for selected tables [676](#)
  - set by AccessMethod=IDheaders [659](#), [661](#)
- ColHead, [Table] WAI keyword [674](#)
  - column extent [674](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- id/header table cell attribute [670](#)
- ColIDs, [Table] WAI keyword [674](#)
  - column extent [674](#)
  - id/header table cell attribute [670](#)
  - override for selected tables [676](#)
- Color\*, [HTMLParaStyles] or [HTMLCharStyles] format property [573](#), [581](#)
- [Colors]
  - map color names to values [440](#)
  - override for paragraph formats [573](#)
- [Colors], specify text colors [580](#)
- ColSpanHead, [Table] WAI keyword [672](#)
  - column-span extent [673](#)
  - id/header table cell attribute [670](#)
- ColSpanIDs, [Table] WAI keyword [672](#)
  - column-span extent [673](#)
  - dependencies [659](#)
  - id/header table cell attribute [670](#)
  - override for selected tables [676](#)
- CombineIndexLevels, [Index] keyword [253](#)
- Comment
  - [HTMLOptions]GeneratorTag setting [435](#)
  - [HTMLParaStyles] format property for scripts [568](#)
  - [HTMLParaStyles] or [HTMLCharStyles] format property [568](#)
- CompactForm, [\*BList] keyword [216](#)
- CompileHelp, [Automation] keyword
  - compile HTML Help project [346](#), [802](#)
- Compiler, [MSHtmlHelpOptions] keyword [346](#)
- ComplexOtherprops, ConditionOptions
  - keyword [162](#)
- CompoundWordChars, [OmniHelpOptions]
  - keyword [372](#)
- ConceptsHead, [RelatedLinksText]
  - keyword [194](#)
- [CondEndFlagAltText], specify flag alt text [168](#)
- [CondEndFlagImages], specify flag images [168](#)
- ConditionalDefaults, ConditionOptions
  - keyword [163](#)
- [ConditionalDefaults], exclude content [164](#)
- [ConditionalExclude], exclude content [164](#)
- [ConditionalFlagging], flag content [163](#)
- [ConditionalInclude], exclude content [164](#)
- [ConditionalPassthrough], pass attributes
  - through [165](#)
- [ConditionAttributes], assign attributes with flags [169](#)
- [ConditionOptions]
  - process otherprops [162](#)
  - set flags [165](#)
  - specify ditaval file [161](#)
- [ConditionOptions], exclude content [163](#)
- [CondStartFlagAltText], specify flag alt text [168](#)
- Config, override configuration settings
  - [HTMLParaStyles] format property [776](#)
  - [MarkerTypes] property [723](#)
- Configs, [Templates] keyword [731](#), [737](#), [742](#), [816](#)
  - precedence of settings [765](#)
- Confluence, [HTMLOptions] keyword [442](#)
- ConfluenceLinkEnd, [HTMLOptions]
  - keyword [442](#)
- ConfluenceLinkPage, [HTMLOptions]
  - keyword [442](#)
- ConfluenceLinkPageEnd, [HTMLOptions]
  - keyword [442](#)
- ConfluenceLinks, [HTMLOptions] keyword [442](#)
- ConfluenceLinkStart, [HTMLOptions]
  - keyword [442](#)
- ConfluenceLinkText, [HTMLOptions]
  - keyword [442](#)
- ConfluenceLinkTextEnd, [HTMLOptions]
  - keyword [442](#)
- Contents
  - [HTMLParaStyles] format property
    - HTML-based Help contents entries [250](#)
  - [JavaHelpOptions] ListType value [249](#), [335](#)
  - [MSHtmlHelpOptions] ListType value [249](#), [335](#)
- [Contents]
  - generate a TOC [221](#)
  - use or lose page break before [525](#)
- ContentsLocalValuePrefix, [MSHtmlHelpOptions] keyword [350](#)
- ContentsNamesFileOnly, [MSHtmlHelpOptions]
  - keyword [338](#)
- [ContentsText]
  - title and other fixed text [199](#)
- [ContentsText], contents entries and links [191](#)

ContentType, [HTMLOptions] keyword 437, 451  
 ContextAnchors, [EclipseHelpOptions]  
     keyword 420, 426  
 ContextDescription, [EclipseHelpOptions]  
     keyword 420, 427  
 ContextFileName, [EclipseHelpOptions]  
     keyword 426  
 ContextID, [EclipseHelpOptions] keyword 426  
 ContextPluginName, [EclipseHelpOptions]  
     keyword 426  
 ContinuedFormatSuffix, [ElementOptions]  
     keyword 89  
 ConversionDPI, [HTMLOptions] keyword 438  
 CopyAfterFiles, [Automation] keyword 795  
 CopyAfterFrom, [Automation] keyword 795  
 CopyBeforeFiles, [Automation] keyword 791  
 CopyBeforeFrom, [Automation] keyword 791  
 CopyCssFrom, [Automation] keyword 800  
     activated by CompileHelp or FTSCCommand 803  
     activated by WrapAndShip 788  
 CopyGraphicsFrom, [Automation] keyword 797  
     activated by CompileHelp or FTSCCommand 803  
     activated by WrapAndShip 788  
     locate graphics for HTML Help 318  
     locate graphics for OmniHelp 360  
 CopyOriginalGraphics, [Automation]  
     keyword 796  
 CousinHead, [RelatedLinksText] keyword 194  
 CshMapFile, [MSHtmlHelpOptions] keyword  
     use symbolic IDs for CSH links 342  
 CshMapFileNumIncrement,  
     [MSHtmlHelpOptions] keyword 342  
 CshMapFileNumStart, [MSHtmlHelpOptions]  
     keyword 342  
 Css\*, [CSS] keywords:  
     CssBodyFontSize 606  
     CssBodyFontTag 607  
     CssBodyFontUnit 607  
     CssBrowserDetect 593  
     CssFileName 593  
         name CSS files 595  
     CssFontUnitDec 607  
     CssFontUnits 607  
     CssIndentBaseSize 607  
     CssIndentBaseUnit 608  
     CssIndentUnitDec 607

CssIndentUnits 607  
 CSSLinkNS4 599  
 CssPath 595  
     destination for CssCopyFiles 795  
     place CSS files for assembly 800  
 [CSS] 592–609  
     file options 593  
     for XML 452  
     link options 547  
     list attributes 588  
 CssCopyFiles, [Automation] keyword 800  
     activated by CompileHelp or FTSCCommand 803  
 [CSSEndMacro], ending code for CSS file 608  
 CSSFlagsFile, ConditionOptions keyword 165  
 CSSReplace, [HTMLParaStyles] or  
     [HTMLCharStyles] format property 608  
 [CSSStartMacro], starting code for CSS file 608  
     specify default font size 607  
     when CSS is generated each time 542  
 CtrlCssName, [OmniHelpOptions] keyword 362

## D

Default, [JavaHelp window] parameter 403  
 DefaultBlockFormat, [ElementOptions]  
     keyword 88  
 DefaultChmFile, [MSHtmlHelpOptions]  
     keyword 317  
     map CHM files 348  
     syntax for inter-CHM-file links 324  
 DefaultInlineFormat, [ElementOptions]  
     keyword 88  
 DefaultNoteType, [ElementOptions]  
     keyword 101  
 DefaultTarget, [HTMLOptions] keyword 444,  
     624  
 DefaultTopic, [JavaHelpOptions] keyword 393  
 DefaultTopic, [OracleHelpOptions]  
     keyword 393  
 DefaultTopicFile  
     [MSHtmlHelpOptions] keyword 318  
     [OmniHelpOptions] keyword 359  
 DefinitionListTables, [TableOptions]  
     keyword 104  
 DefListTableColWidths, [TableOptions]  
     keyword 104



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

DefListTableFormat, [TableOptions]  
 keyword [104](#)

Delete  
 [HTMLCharStyles] format property, override  
 placement [772](#)  
 [HTMLParaStyles] format property [666](#), [784](#)  
 delimit extracts [530](#)  
 do not use for CodeStore paragraphs [694](#)  
 eliminate glossary entries from JavaHelp  
 TOC [403](#)  
 eliminate unwanted paragraphs [569](#)  
 enable/disable extract processing [528](#)  
 for configuration overrides [776](#)  
 hide TextStore paragraphs [693](#)  
 hide WAI markup [650](#), [651](#), [652](#)  
 HTML Help TOC-only entries [337](#)  
 name split files [784](#)  
 override placement [772](#)  
 paragraph formats for <meta> tags [437](#)  
 prevent duplicate file names [785](#)  
 [MarkerTypes] property [723](#)  
 [XrefStyles] format property [552](#)

DeleteExistingDCL, [Setup] keyword [73](#)

DescendantHead, [RelatedLinksText]  
 keyword [194](#)

DescriptionIsFirstLabel,  
 [EclipseHelpOptions] keyword [420](#), [426](#)

DisableChunking, [Chunking] keyword [524](#)

DisplayElementPath, [ElementOptions]  
 keyword [73](#)

DitavalFile, ConditionOptions keyword [161](#)

DListDD, [HTMLParaStyles] format property [586](#),  
[588](#)

Document, [Templates] keyword [732](#), [737](#), [738](#)

Drop\*, [DropDowns] keywords  
 DropBlockEnd [271](#)  
 DropBlockStart [271](#)  
 DropButton [271](#)  
 DropButtonAttr [271](#)  
 DropButtonCloseLabel [270](#)  
 DropButtonOpenLabel [270](#)  
 DropClass [271](#)  
 DropCloseIcon [271](#)  
 DropCloseIconAlt [269](#)  
 DropCloseIconFile [269](#)  
 DropDivAttr [271](#)  
 DropDownBlock [266](#)  
 DropIDPrefix [270](#)

DropJSCode [272](#)  
 DropJSLocation [272](#)  
 DropLinkAttr [270](#)  
 DropLinkEnd [270](#)  
 DropLinkPara [271](#)  
 DropLinkParaEnd [270](#)  
 DropLinkParaStart [270](#)  
 DropLinkParaText [270](#)  
 DropLinkStart [270](#)  
 DropLinkType [268](#)  
 DropOpenIcon [271](#)  
 DropOpenIconAlt [269](#)  
 DropOpenIconFile [269](#)  
 DropText [270](#)

DropDown, [HTMLParaStyles] format  
 property [265](#), [266](#)

DropDownEnd, [HTMLParaStyles] format  
 property [266](#)

DropDownLink, [HTMLParaStyles] format  
 property [266](#)

DropDownLink, [HtmlParaStyles] or  
 [HtmlCharStyles] format property [266](#)

[DropDowns], create expandable drop-down  
 sections [268](#)

DropDownStart, [HtmlCharStyles] format  
 property [266](#)

DropDownStart, [HTMLParaStyles] format  
 property [266](#)

DTDPPath, [Setup] keyword [33](#), [69](#)

DuplicateNameCheck, [IDOptions] keyword [77](#)

## E

EclipseVer, [EclipseHelpOptions]  
 keyword [415](#)

EditorFileName, [Logging] keyword [74](#)

[ElementAttrPrefixes], prefix format names  
 based on attributes [95](#)

[ElementClasses], add ^class to elements [183](#)

[ElementOptions], assign formats to elements [88](#)

ElementPathFormat, [ElementOptions]  
 keyword [73](#)

[ElementSets], define sets of elements [179](#)

[ElementText], miscellaneous text  
 assignments [159](#)

[ElementTypes], properties of element types [183](#)



EmptyGraphPath, [Automation] keyword 798  
     activated by CompileHelp or FTSCCommand 803  
     when effective 793

EmptyJavaGraphSubdir, [JavaHelpOptions]  
     keyword 390

EmptyJavaGraphSubdir, [OracleHelpOptions]  
     keyword 390

EmptyJavaHTMLSubdir, [JavaHelpOptions]  
     keyword 390

EmptyJavaHTMLSubdir, [OracleHelpOptions]  
     keyword 390

EmptyOutputDir, [Automation] keyword 789  
     activated by CompileHelp or FTSCCommand 803  
     dependencies 791  
     when effective 790

EmptyOutputFiles, [Automation] keyword 790  
     activated by CompileHelp or FTSCCommand 803  
     when to include 791

EmptyParaContent, [HTMLOptions] keyword 569

EmptyTbCellContent, [Table] keyword 640  
     eliminate default content for XML 453

EmptyWrapPath, [Automation] keyword 793  
     activated by CompileHelp or FTSCCommand 803  
     dependencies 799

Encoding, [HTMLOptions] keyword 434  
     for XML 450  
     prevent character mapping 576

End, [Inserts] keyword 535, 536, 710  
     for framesets 444

[End], dummy section to end settings 66

[End], dummy section to replace  
     [MacroVariables] 680

EndingFSButton, [NavigationMacros]  
     keyword 563

EndingNextFSButton, [NavigationMacros]  
     keyword 562

EndingNextFSMacro, [NavigationMacros]  
     keyword 561  
     scope 562

Entities, [Inserts] keyword 432, 710

Extr\*  
     [Graphics] extract keywords:  
         ExtrGraphClass 541  
         ExtrGraphHigh 541  
         ExtrGraphSuffix 537, 539, 540  
         ExtrGraphTarget 541

ExtrGraphThumbnail 540  
     ExtrGraphWide 541

[HTMLParaStyles] extract format properties  
     ExtrDisable 529  
     ExtrEnable 529  
     ExtrEnd 529  
     ExtrFinish 529  
     ExtrStart 529

[Inserts] keywords:  
     ExtrBottom 536  
     ExtrHead 536  
     ExtrHeadEnd 536  
     ExtrTop 536

[MarkerTypes] properties  
     ExtrBottom 723  
     ExtrDisable 723  
     ExtrEnable 723  
     ExtrEnd 724  
     ExtrFinish 724  
     ExtrHead 724  
     ExtrReplace 724  
     ExtrStart 724  
     ExtrTop 724

[Extr\*], extract-file sections  
     *all subject to configuration overrides* 772  
     [ExtrBottom] 536  
     [ExtrHead] 536  
     [ExtrReplace], extract replacement code 539  
     [ExtrTitle] 538  
     [ExtrTop] 536

ExtractEnable, [HTMLOptions] keyword 528

## F

FigTitleAboveImage, [FigureOptions]  
     keyword 102

FigTitleAboveTable, [FigureOptions]  
     keyword 102

Figure, [HTMLParaStyles] format property, ensure wrapping DITA image in <fig> 483

FigureAnchorFormat, [FigureOptions]  
     keyword 103

[FigureOptions] 102

FileName  
     [HTMLParaStyles] format property 783  
     [MarkerTypes] property 724  
     name split and extract files 782

FileName. [\*BList] keyword 215

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

FileNameSpaceChar, [HTMLOptions]  
     keyword [783](#)  
 FileSuffix, [Setup] keyword [71](#), [450](#)  
     determined at run time [733](#)  
     for DITA output [456](#)  
     for DocBook output [501](#)  
 First\*, [Inserts] keywords:  
     FirstBottom [536](#)  
     FirstEnd [536](#)  
     FirstFrames [536](#)  
     FirstHead [536](#)  
     FirstHeadEnd [536](#)  
     FirstTop [536](#)  
 FixGraphSpaces, [Graphics] keyword [748](#)  
 Font, [Base] keyword [577](#)  
 [FontSizes], map points to HTML sizes [577](#)  
 FootClass, [CSS] keyword [603](#)  
 FootInlineIDPrefix, [HTMLOptions]  
     keyword [503](#), [582](#)  
 FootInlineParaTag, [HTMLOptions]  
     keyword [503](#), [582](#)  
 FootInlineRefTag, [HTMLOptions]  
     keyword [503](#), [582](#)  
 FootInlineTag, [HTMLOptions] keyword [582](#)  
 FootnoteEndCode, [HTMLOptions] keyword [583](#)  
 FootnoteFormat, [Footnotes] keyword [102](#)  
 Footnotes, [HTMLOptions] keyword [503](#), [582](#)  
     XML default value [449](#)  
 [Footnotes], assign footnote formats [102](#)  
 FootnoteSeparator, [HTMLOptions]  
     keyword [582](#)  
 FootnoteStartCode, [HTMLOptions]  
     keyword [583](#)  
 FootnoteStream, [NumberStreams] keyword [147](#)  
 FootTagLast, [Table] keyword [630](#)  
 [FormatAliases]  
     map format names [52](#), [113](#)  
     map subformat names [142](#)  
 Formats, [Templates] keyword [110](#), [120](#), [121](#), [730](#)  
 FrameHigh, [OmniHelpOptions] keyword [363](#)  
 FrameOptions, [OmniHelpOptions] keyword [363](#)  
 Frames, [Inserts] keyword [535](#), [536](#), [710](#)  
     for framesets [444](#)  
 Frameset, [OmniHelpOptions] keyword [363](#)  
 FrameWide, [OmniHelpOptions] keyword [363](#)

FriendHead, [RelatedLinksText] keyword [194](#)  
 FRowsN, [TableAccess] keyword [631](#), [632](#)  
 FTSCCommand, [JavaHelpOptions] keyword  
     for JavaHelp [398](#)  
 FTSCCommand, racleHelpOptions] keyword  
     for Oracle Help [399](#)  
 FullIndexRanges, [Index] keyword [209](#)

## G

GenerateALinks, [RelatedLinks] keyword [192](#)  
 GenerateIDX, [Index] keyword [222](#)  
 GenerateMapIfMissing, [MapGeneration]  
     keyword [69](#)  
 GenerateParentChild, [RelatedLinks]  
     keyword [190](#)  
 GeneratePrevNext, [RelatedLinks]  
     keyword [190](#)  
 GenerateSiblings, [RelatedLinks]  
     keyword [190](#)  
 GenerateTOC, [Contents] keyword [221](#)  
 GenerateUIDs, [IDOptions] keyword [76](#)  
 GeneratorTag, [HTMLOptions] keyword [435](#)  
 GenListXrefFormat, [ElementOptions]  
     keyword [102](#)  
 [Glossary], define glossary output [204](#)  
 [GlossaryText], text for title of generated  
     glossary [204](#)  
 GlossPrefix, [JavaHelpOptions] keyword [402](#)  
 GlossSpace, [JavaHelpOptions] keyword [402](#)  
 GlossSuffix, [JavaHelpOptions] keyword [402](#)  
 GlossTerm, [HTMLParaStyles] JavaHelp format  
     property [402](#)  
 GLSFile, [Glossary] keyword [204](#)  
 GLSSuffix, [Glossary] keyword [204](#)  
 GLSTitle, [GlossaryText] keyword [204](#)  
 GLSTitleFormat, [Glossary] keyword [204](#)  
 GLSTOCFormat, [Glossary] keyword [204](#)  
 [GraphALT], image alt attribute  
     for image maps [622](#)  
 [GraphALT], image alt tags [620](#)  
     subject to configuration overrides [775](#)  
 [GraphAttr], image attributes [619](#)

- subject to configuration overrides [775](#)
- GraphClass, [Graphics] keyword [601](#)
- GraphCopyFiles, [Automation] keyword [797](#)
  - activated by CompileHelp or FTSCCommand [803](#)
- [GraphEndMacros], code after images [615](#)
  - subject to configuration overrides [775](#)
- [GraphFiles], replace graphics [613](#), [747](#), [799](#)
  - path overrides [748](#)
  - subject to configuration overrides [775](#)
- [GraphGroup], create graphics groups [615](#)
  - assign with **\*Config** marker [775](#)
- [GraphHigh], image height in pixels [621](#)
  - property of extracted graphic [542](#)
  - related to predefined macro variables [616](#)
  - subject to configuration overrides [775](#)
- GraphicAlignment, [FigureOptions] keyword [103](#)
- [Graphics]
  - class name for anchor paragraph [601](#)
  - fix graphics file names [748](#)
  - graphics location for JavaHelp [391](#)
  - relocate graphics files [70](#), [611](#), [747](#)
  - remove path information [347](#)
  - replace graphics [747](#), [799](#)
  - subject to configuration overrides [770](#)
  - third-party graphics tools [78](#)
  - thumbnails for extract links [540](#)
  - use existing graphics files [612](#), [748](#)
  - use title for alt [623](#)
  - include or omit image attributes
    - for DITA XML [484](#)
    - for DocBook XML [521](#)
    - for generic XML [453](#)
    - for HTML [621](#)
- [GraphIndents], indent images [618](#)
- GraphPath, [Graphics] keyword [70](#), [611](#), [747](#), [799](#)
  - for JavaHelp and Oracle Help [391](#)
  - overridden by [GraphFiles] [747](#)
  - overrides [GraphFiles] [748](#)
- GraphPathOverrides, [Graphics] keyword [748](#), [774](#), [799](#)
- [GraphReplaceMacros], code instead of image [615](#)
  - subject to configuration overrides [775](#)
- [GraphRightSpacers], indent images [618](#)
- GraphScale, [Graphics] keyword
  - eliminate attributes
    - for DITA XML [484](#)
    - for DocBook XML [521](#)
    - for generic XML [453](#)
    - for HTML [621](#)
  - XML default value [449](#)
- [GraphScale], scale images [621](#)
  - related to predefined macro variables [616](#)
  - subject to configuration overrides [775](#)
- [GraphStartMacros] [615](#)
  - subject to configuration overrides [775](#)
- GraphSubdir, [JavaHelpOptions] keyword [390](#)
- GraphSubdir, [OracleHelpOptions] keyword [390](#)
- GraphSuffix, [Graphics] keyword [747](#), [799](#)
  - replace referenced graphics [613](#)
  - third-party graphics tools [78](#)
  - use referenced graphics without converting [612](#)
- [GraphSuffix], replace graphics file extension [613](#), [747](#), [799](#)
- [GraphWide], width of image in pixels [621](#)
  - property of extracted graphics [542](#)
  - related to predefined macro variables [616](#)
  - subject to configuration overrides [775](#)
- GraphWrapPara, [Graphics] keyword [453](#), [616](#)

## H

- h1 - h6, [ParaTags] format properties [566](#)
- HColSN, [TableAccess] keyword [631](#), [632](#)
  - default header columns [631](#)
  - effect on [Table]ScopeRow [668](#)
- Head, [Inserts] keyword [535](#), [710](#)
  - customize CSS link tag [599](#)
  - for CSS selection macro [594](#), [597](#), [598](#)
  - for HTML Help KeyHelp pop-up [322](#)
  - for solitary file [536](#)
- HeadBlockFormat, [RelatedLinks] keyword [194](#)
- HeadEnd, [Inserts] keyword [535](#)
  - for solitary file [536](#)
- HeadFootBodyTags, [Table] keyword [629](#)
  - choose a row-group method [672](#)
  - default header/footer counts [631](#)
  - enable [Table\*Attributes] [644](#)
  - identify table cells via scope [668](#)
  - RowGroup property [677](#)
  - with scope method [658](#), [661](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- HeadInlineFormat, [RelatedLinks]
  - keyword [194](#)
- Height, [JavaHelp window] parameter [404](#)
- Helen, [JavaHelpOptions] keyword [395](#)
- [HelpContentsLevels]
  - assign heading levels for split overrides [527](#)
  - for HTML-based Help [250](#), [251](#)
  - subject to configuration overrides [772](#)
- HelpFileLanguage, [MSHtmlHelpOptions]
  - keyword [345](#)
- HelpFileName
  - [JavaHelpOptions] keyword [393](#)
  - [MSHtmlHelpOptions] keyword [316](#)
  - [OmniHelpOptions] keyword [359](#)
  - [OracleHelpOptions] keyword [393](#)
- HelpMerge, [MarkerTypes] property [724](#)
- [HelpMerge], merge help projects
  - for HTML Help [350](#)
  - for OmniHelp [378](#), [379](#)
- [HelpMergePaths], merge JavaHelp or Oracle Help helpsets [410](#)
- HFBTags, [TableAccess] override [630](#), [644](#)
- HHCPProperties, [MSHtmlHelpOptions]
  - keyword [339](#)
- HHKProperties, [MSHtmlHelpOptions]
  - keyword [339](#)
- HHPFileName, [MSHtmlHelpOptions]
  - keyword [317](#)
  - archive deliverables [806](#)
- [HHWindows], secondary windows
  - for HTML Help [333](#)
- HistoryFileName, [Logging] keyword [74](#)
- HrefAttribute, [HTMLOptions] keyword [454](#), [502](#)
- HRowsN, [TableAccess] keyword [631](#), [632](#)
  - default header row count [631](#)
  - effect on [Table]ScopeCol [668](#)
- HSFileName, [JavaHelpOptions] keyword [401](#)
  - archive JavaHelp deliverables [806](#)
  - name JavaHelp helpset file [392](#)
- HSFileName, [OracleHelpOptions] keyword
  - name Oracle Help helpset file [392](#)
- HSPathNames, [JavaHelpOptions] keyword [394](#)
- HSPathNames, [OracleHelpOptions]
  - keyword [394](#)
- HTMConfig
  - [HTMLParaStyles] format property [776](#)
  - [MarkerTypes] property [724](#)
- [HTMLCharStyles]
  - subject to configuration overrides [772](#)
- HTMLComment, [MarkerTypes] property [724](#)
- HTMLDocType, [HTMLOptions] keyword [432](#)
  - for framesets [444](#)
- HTMLDTD, [HTMLOptions] keyword [433](#)
  - for framesets [444](#)
- [HtmlFiles], rename split files [782](#)
- [HTMLOptions]
  - for declarations [432](#), [433](#), [434](#), [437](#), [438](#), [444](#)
  - for extracts [528](#)
  - for footnotes [503](#), [582](#)
  - for framesets [444](#)
  - for links [444](#), [624](#)
  - for preformatted text [439](#), [581](#)
  - for split files [527](#), [783](#)
  - for tables [641](#)
  - subject to configuration overrides [770](#)
- [HTMLParaStyles]
  - for extracts [529](#)
  - for HTML Help [261](#)
  - for images [651](#)
  - for links [652](#)
  - for split files [532](#)
  - for WAI table attributes [662](#)
  - subject to configuration overrides [772](#)
- HTMLSubdir, [JavaHelpOptions] keyword [390](#), [392](#)
- HTMLSubdir, [OracleHelpOptions] keyword [390](#)
- HTMLVersion, [HTMLOptions] keyword [432](#)
- HyperSpaceChar, [HTMLOptions] keyword [549](#)

## I

- IDElemSep, [IDOptions] keyword [77](#)
- IDFile, [IDOptions] keyword [76](#)
- IDheaders, [Table]AccessMethod option [658](#)
  - apply id/headers to all tables [659](#)
  - ColGroup dependency [676](#)
  - RowGroup dependency [677](#)
- [IDOptions] options for element IDs [76](#)
- IDPath, [IDOptions] keyword [76](#)
- IDPathSep, [IDOptions] keyword [76](#)

- IDs, [TableAccess] property, [659](#)
- IDTopic, [IDOptions] keyword [76](#)
- IDTopSep, [IDOptions] keyword [77](#)
- IDUpDir, [IDOptions] keyword [77](#)
- IdxButtons, [OmniHelpOptions] keyword [369](#)
- IdxExpand, [OmniHelpOptions] keyword [368](#)
- IDXFile, [Index] keyword [212](#)
- IdxFilename, [EclipseHelpOptions]  
keyword [419](#)
- IDXFormat, [Index] keyword [207](#)
- IdxGroupsOpen, [OmniHelpOptions]  
keyword [369](#)
- IdxIcoBase, [OmniHelpOptions] keyword [369](#)
- IdxOpenLevel, [OmniHelpOptions] keyword [369](#)
- IDXSuffix, [Index] keyword [212](#)
- IDXTitle, [IndexText] keyword [191](#), [207](#)
- IDXTitleFormat, [Index] keyword [207](#)
- IDXTOCFormat, [Index] keyword [207](#)
- IECssName, [OmniHelpOptions] keyword [361](#)
- IECtrlCssName, [OmniHelpOptions]  
keyword [362](#)
- IgnoreCharsIX, [Index] keyword [256](#)
- IgnoreLeadingCharsIX, [Index] keyword [256](#)
- Image, [*JavaHelp* window] parameter [404](#)
- [ImportancePrefixes], format-name prefixes  
based on importance value [96](#)
- IncludeElementTags, [ElementOptions]  
keyword [72](#)
- IncludeVersionPI, [EclipseHelpOptions]  
keyword [418](#)
- Index
  - [JavaHelpOptions]ListType value [249](#), [335](#)
  - [MShtmlHelpOptions]ListType value [249](#),  
[335](#)
- [Index]
  - configure Help index entries [253](#)
  - configure HTML index entries [211](#)
  - generate an index [222](#)
  - file name and suffix [212](#)
- IndexBottom, [Inserts] keyword [213](#)
- [IndexClasses], map indexterms to indexlist  
variants [218](#)
- IndexFileSuffix, [Index] keyword [212](#)
- IndexHead, [Inserts] keyword [213](#)
- IndexLetterClass, [Index] keyword [213](#)
- IndexLetterNumber, [Index] keyword [210](#)
- IndexLetterPrefix, [Index] keyword [210](#)
- IndexLettersFormat, [Index] keyword [210](#)
- IndexLetterSymbol, [Index] keyword [210](#)
- IndexLevelClass, [Index] keyword [212](#)
- [IndexLists], assign indexterm outputclasses to in-  
dexlist variants [218](#)
- IndexNavType, [Index] keyword [211](#)
- IndexRangeSep, [IndexText] keyword [210](#)
- IndexRefClass, [Index] keyword [212](#)
- IndexRefSep, [IndexText] keyword [210](#)
- IndexRefStartSep, [IndexText] keyword [210](#)
- IndexSeeAlsoEnd, IndexSeeFormats]  
keyword [208](#)
- IndexSeeAlsoStart, IndexSeeFormats]  
keyword [208](#)
- IndexSeeEnd, IndexSeeFormats] keyword [208](#)
- [IndexSeeFormats], assign index see and see-also  
formats [208](#)
- IndexSeeStart, IndexSeeFormats] keyword [208](#)
- [IndexSeeText], index see/see-also entries [209](#)
- IndexSortLocale, [HtmlOptions] keyword [257](#)
- IndexSortType, [HtmlOptions] keyword [257](#)
- [IndexText], index entries and links [191](#)
- [IndexText], text of index title [207](#)
- IndexTOC, [Inserts] keyword [213](#)
- IndexTop, [Inserts] keyword [213](#)
- IndexTopLettersClass, [Index] keyword [213](#)
- IndexTopLettersFormat, [Index] keyword [211](#)
- [InlineFormatMaps], map inline element paths to  
formats [93](#)
- [InlineFormatPrefixRunins], specify run-in  
headings [98](#)
- [InlineOutclassMaps], assign format names to in-  
line elements [90](#)
- [InlineOutclassMaps], for <xref> wrappers [91](#)
- [Inserts], insert code at predefined locations [535](#),  
[710](#)  
for HTML indexes [213](#)  
subject to configuration overrides [771](#)
- InternalTableCaption, [Table] keyword [503](#),  
[641](#)



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

`Ital`, [HTMLPararStyles] or [HTMLCharStyles]  
format property 573

`ItemFormat`. [\*BList] keyword 215

## J

`JarCommand`, [JavaHelpOptions] keyword 401

`JarCommand`, [OracleHelpOptions] keyword 401

[*JavaHelp window*], assign default parameters 403

[JavaHelpOptions]  
subject to configuration overrides 771

`JavaRootFiles`, [JavaHelpOptions]  
keyword 390

`JavaRootFiles`, [OracleHelpOptions]  
keyword 390

[JH2\_HelpsetAddition] 394

[JHImages] 404

`JHVersion2`, [JavaHelpOptions] keyword 387

## K

`KeepDraftComments`, [ElementOptions]  
keyword 105

`KeepFileNameSpaces`, [HTMLOptions]  
keyword 783

`KeepFileNameUnderscores`, [HTMLOptions]  
keyword 783

`KeepTOCWithTitlePage`, [Contents]  
keyword 525

`KeydefsOnlyWithinBranch`, KeyOptions  
keyword 172

[KeyOptions], limit scope of keydefs 172

`KeywordLimit`, [MShtmlHelpOptions]  
keyword 252

`KeywordRefs`, [Index] keyword 255

`KLink*`, [MShtmlHelpOptions] keywords:

- `KLinkButtonGraphic` 326
- `KLinkButtonHeight` 326
- `KLinkButtonIcon` 326
- `KLinkButtonText` 326
- `KLinkButtonWidth` 326
- `KLinkEmptyTopic` 326
- `KLinkFlags` 326
- `KLinkText` 326
- `KLinkTextFont` 326

`KLinkType` 326

## L

Languages, [Templates] keyword 158, 730, 740

`Last*`, [Inserts] keywords:

- `LastBottom` 536
- position trails of links 559
- `LastEnd` 536
- `LastFrames` 536
- `LastHead` 536
- `LastHeadEnd` 536
- `LastTop` 536

`Left`

- [HTMLParaStyles] format property 573
- [*JavaHelp window*] parameter 404

`LeftWide`, [OmniHelpOptions] keyword 363

`LEnd`, [HTMLParaStyles] format property 586, 587

`LevelBreakForSee`, [Index] keyword 255

`LFirst`, [HTMLParaStyles] format property  
for lists with multiple paragraph formats 587  
start list style 586

`Link*`, [MShtmlHelpOptions] base keywords:

- `LinkButtonGraphic` 326
- `LinkButtonHeight` 326
- `LinkButtonIcon` 326
- `LinkButtonText` 326
- `LinkButtonWidth` 326
- `LinkEmptyTopic` 326
- `LinkFlags` 326
- `LinkText` 326
- `LinkTextFont` 326
- `LinkType` 326

`LinkClass`, [HTMLParaStyles] WAI keyword 652

`LinkClassIsParaClass`, [CSS] keyword 547  
default depends on UseCSS 597

`LinkFormat`, [ElementOptions] keyword 89

`LinkSrc`

- [HTMLCharStyles] format property 712
- [HTMLParaStyles] format property 548, 712
- [XrefStyles] format property 552

`LinkTitle`, [HTMLParaStyles] WAI keyword 652

`ListMissingRefs`, [HTMLOptions] keyword 551

`ListN`, [HTMLParaStyles] format property 586

[ListOfFiguresText], text for generated LOF 201

[ListOfTables], define LOT output 202

[ListOfTablesText], text for generated LOT 202

[ListOptions], list styles 127

ListStream, [NumberStreams] keyword 147

ListTitle, [\*BLText] keyword 216

ListType

- [EclipseHelpOptions] keyword 420
- [JavaHelpOptions] keyword 249, 335
- [MShtmlHelpOptions] keyword 249, 335
- [OmniHelpOptions] keyword 367

LLevel, [HTMLParaStyles] format property 586, 587

LNest, [HTMLParaStyles] format property 586, 587

LockAllNavtitles, [MapOptions] keyword 200

LOFFile, [ListOfFigures] keyword 201

LOFFormat, [ListOfFigures] keyword 202

LOFSuffix, [ListOfFigures] keyword 201

LOFTitle, [ListOfFiguresText] keyword 201

LOFTitleFormat, [ListOfFigures] keyword 201

LOFTOCFormat, [ListOfFigures] keyword 202

LOFXrefFormat, [ListOfFigures] keyword 202

LogAuto, [Automation] keyword 788

LogDebug, [Logging] keyword 74

LogErrors, [Logging] keyword 74

LogFileName, [Logging] keyword 74

[Logging] conversion events 74

LogInfo, [Logging] keyword 74

LogIniChains, [Logging] keyword 75

LogQuerys, [Logging] keyword 74

LogWarnings, [Logging] keyword 74

Longdesc, [HTMLParaStyles] format property for longdesc attribute 651

LOTFile, [ListOfTables] keyword 202

LOTFormat, [ListOfTables] keyword 203

LOTSuffix, [Contents] keyword 202

LOTTitle, [ListOfTablesText] keyword 202

LOTTitleFormat, [ListOfTables] keyword 202

LOTTOCFormat, [ListOfTables] keyword 203

LOTXrefFormat, [ListOfTables] keyword 203

LowerCaseCSS, [CSS] keyword 600

LowMem, [OmniHelpOptions] keyword 359

## M

MacroNestMax, [Macros] keyword 683, 705

Macros, [Templates] keyword 684, 729

[Macros]

- debug 709
- loop-control limits 705
- remove implicit line breaks 681
- subject to configuration overrides 771

[MacroTemplates], assign a template macro to a file 187

[MacroVariables] 689

- create a macro variable 688

MacroVarNesting, [Macros] keyword 690

MainCssName, [OmniHelpOptions] keyword 361

MakeAliasFile, [MShtmlHelpOptions] keyword 343

MakeALinkFile, [OracleHelpOptions] keyword 409

MakeArchive, [Automation] keyword 43

MakeCshMapFile, [MShtmlHelpOptions] keyword 342

MakeFileHrefsLower, [HTMLOptions] keyword 415, 549

- for JavaHelp 401

MakeFTS, [Automation] keyword 43

MakeJar, [Automation] keyword 43

MakeTrail, [Trails] keyword 556

- enable [HTMLParaStyles]Trail format property 556

MapFilePrefix, [JavaHelpOptions] keyword 392

MapFilePrefix, [OracleHelpOptions] keyword 392

[MapGeneration], produce DITA map from topics 69

MapLanguage, [MapGeneration] keyword 70

[MapOptions], configure generated maps 106, 200

MapRootElement, [MapGeneration] keyword 69

MapTitle, [MapGeneration] keyword 69

MapTitleElem, [MapGeneration] keyword 69

[MarkerTypeCodeAfter] 725

[MarkerTypeCodeBefore] 724

[MarkerTypeCodeReplace] 725



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

[MarkerTypes], marker-type properties 723  
 MergeFirst, [OmniHelpOptions] keyword 379  
 MergePre, [HTMLOptions] keyword 439  
 Meta  
     [HTMLOptions]GeneratorTag property 435  
     [HTMLParaStyles] format property 436, 534  
 MidHigh, [OmniHelpOptions] keyword 363  
 MoveArchive, [Automation] keyword 807  
 [MSHtmlHelpOptions] 316–352  
     subject to configuration overrides 771

## N

N4CssName, [OmniHelpOptions] keyword 361  
 N4CtrlCssName, [OmniHelpOptions]  
     keyword 362  
 N6CssName, [OmniHelpOptions] keyword 361  
 N6CtrlCssName, [OmniHelpOptions]  
     keyword 362  
 Name, [JavaHelp window] parameter 404  
 NameUndefinedMacros, [Macros] keyword 709  
 NameUndefinedMacroVars, [Macros]  
     keyword 709  
 NavElems, [OmniHelpOptions] keyword 367  
 NavIcons, [JavaHelp window] parameter 404  
 [NavigationMacros]  
     use buttons for 560  
     559  
 NavPane, [JavaHelp window] parameter 404  
 NewWindow, [OmniHelpOptions] keyword 363  
 NextButton, [NavigationMacros] keyword 562  
 NextHead, [RelatedLinksText] keyword 194  
 NextMacro, [NavigationMacros] keyword 561  
 NoAccess, [TableAccess] property 659  
 NoAnum, [HTMLParaStyles] format property, re-  
     move autonumbers 454, 567  
     from footnotes 584  
 NoAttribLists, [CSS] keyword 588  
 NoClassLists, [CSS] keyword 588  
     default depends on UseCSS 597  
 NoColID, [HTMLParaStyles] WAI format  
     property 662  
 NoColor, [HTMLParaStyles] or

    [HTMLCharStyles] format property 573, 581  
 NoContLink, [HTMLParaStyles], HTML Help for-  
     mat property 337, 338  
 NoCSS, [HTMLParaStyles] or [HTMLCharStyles]  
     format property 608  
 NoFig, [HTMLParaStyles] or [HTMLCharStyles]  
     format property, prevent wrapping DITA image  
     in <fig> 483  
 NoFonts, [HTMLOptions] keyword 578  
     CSS-dependent default value 597  
     prevent <font> tags from overriding CSS 609  
     XML default value 449  
 NoFootnoteLinks, [HTMLOptions] keyword 583  
 NoLocations, [HTMLOptions] keyword 549  
 NoMemDel, [Options] keyword 821  
 NoNameDel, [Options] keyword 821  
 NonsplitBottom, [Inserts] keyword 536  
 NonsplitTop, [Inserts] keyword 536  
 NoPara, [HTMLParaStyles] format property, strip  
     <p> tags 568  
     for XML 453  
 NoParaClose, [HTMLOptions] keyword 438  
 NoRef  
     [HTMLParaStyles] format property 422, 549  
     [XrefStyles] format property 552  
 NormalTableFormat, [TableOptions]  
     keyword 103  
 NoSize, [HTMLParaStyles] or  
     [HTMLCharStyles] format property 573  
 NoSplit  
     [HTMLParaStyles] format property 528  
 NoTags, [HTMLParaStyles] or  
     [HTMLCharStyles] format property 568  
 [NoteAttrPrefixes], attributes to determine which  
     Note format to use 96  
 NoWrap, [HTMLOptions] keyword 439, 451  
 NoWrap, [HTMLParaStyles] format property, sup-  
     press line breaks 568  
 [NumberFormatsText], text for numbering  
     formats 149  
 NumericCharRefs, [HTMLOptions] keyword 434  
     for XML 450

## O

OHProjFileSuffix, [OmniHelpOptions]  
keyword 358

OHProjFileXhtml, [OmniHelpOptions]  
keyword 354

[OHTopLeftNav], code for OmniHelp 364

[OHTopRightNav], code for OmniHelp 364

OHVFiles, [OmniHelpOptions] keyword 380

OHViewPath, [OmniHelpOptions] keyword 355,  
380

OmitMacroReturns, [Macros] keyword  
omit line breaks in macro output 681

[OmniHelpOptions] 358–379  
subject to configuration overrides 771

OnlyAuto, [Automation] keyword 808

[Options]  
for cases, spaces, and wildcards 73  
for conversion debugging 821  
subject to configuration overrides 771

[OracleHelpWindows] 408

[OrderedListElement], implied list format 126

OutputclassHasBorderShadeFormats,  
[ElementOptions] keyword 88

OutputclassHasBorderShadeFormatst,  
[ElementOptions] keyword 145

## P

Padding, [Table] keyword 636  
overridden by [TableAttributes] 633, 637

Pages, [Setup] keyword 134

Pages, [Templates] keyword 110

[ParaClasses]  
default use for CSS class names 592  
map paragraph formats 601  
map paragraph formats for XML 452  
subject to configuration overrides 771

ParaLinkClass, [HTMLParaStyles] format  
property 547

ParameterListTables, [TableOptions]  
keyword 104

ParamListTableColWidths, [TableOptions]  
keyword 104

ParamListTableFormat, [TableOptions]

keyword 104

[Parar], default paragraph format 122

[ParaStyle\*] sections  
[ParaStyleCode\*] sections  
*all subject to configuration overrides* 772  
[ParaStyleCodeAfter] 712  
[ParaStyleCodeBefore] 550, 712  
[ParaStyleCodeEnd] 712  
[ParaStyleCodeReplace] 712  
[ParaStyleCodeStart] 691, 712  
[ParaStyleCSS] 608  
[ParaStyleLinkSrc] 548, 712

[ParaTags]  
assign HTML tags to paragraph formats 566  
assign XML tags to paragraph formats 452  
designate script paragraph formats 568  
map format names to CSS class names 601  
subject to configuration overrides 772  
tags used for CSS classes by default 592

ParentHead, [RelatedLinksText] keyword 194

PartStream, [NumberStreams] keyword 147

[PeerLinks], resolve peer related links 196

PersistSettings, [OmniHelpOptions]  
keyword 365

PixelSpacerImage, [HTMLOptions] keyword 618  
indent images 618  
indent tables 641

Plain, [HTMLParaStyles] or [HTMLCharStyles]  
format property 573

PluginID, [EclipseHelpOptions] keyword 417

PluginName, [EclipseHelpOptions]  
keyword 416

PluginProvider, [EclipseHelpOptions]  
keyword 417

PluginSchemaVersion, [EclipseHelpOptions]  
keyword 419

PluginVer, [EclipseHelpOptions]  
keyword 417

Pop\*  
[JavaHelpOptions] keywords:  
PopFontColor 406  
PopFontFamily 406  
PopFontSize 406  
PopFontStyle 406  
PopFontWeight 406  
PopGraphic 406  
PopMarkerPrefix 407

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

PopSize [405](#)  
 PopText [406](#)  
 PopType [406](#)  
 [MSHtmlHelpOptions] keywords:  
   PopColors [322](#)  
   PopFont [322](#)  
   PopMargins [322](#)

pre, [ParaTags] format property [566](#)  
 PrevButton, [NavigationMacros] keyword [562](#)  
 PrevHead, [RelatedLinksText] keyword [194](#)  
 PrevMacro, [NavigationMacros] keyword [561](#)  
 PrintProject, [Setup] keyword [71](#)  
 ProjectName, [OmniHelpOptions] keyword [358](#)  
 ProjectTemplate, [OmniHelpOptions]  
   keyword [366](#)  
 PropertiesTableFormat, [TableOptions]  
   keyword [103](#)  
 PublicID, [MapGeneration] keyword [70](#)

## Q

QuotedEncoding, [HTMLOptions] keyword [434](#)  
 Quotes, ElementText] keyword [143](#), [159](#)

## R

Raw, [HTMLParaStyles] format property  
   for ALink paragraphs [329](#)  
   for marker-only paragraphs [724](#)  
   for split files [532](#)  
 Raw, [HTMLParaStyles] or [HTMLCharStyles]  
   format property [568](#)  
 ReferenceFlagsFile, ConditionOptions  
   keyword [165](#)  
 ReferencesHead, [RelatedLinksText]  
   keyword [194](#)  
 RefFileType  
   [JavaHelpOptions] keyword [249](#)  
   [MSHtmlHelpOptions] keyword [249](#), [335](#)  
   [OmniHelpOptions] keyword [249](#), [367](#)  
 RefFormat. [\*BList] keyword [216](#)  
 RefID. [\*BList] keyword [215](#)  
 [RegMark], default registered-trademark format [157](#)  
 RelatedDividerFormat, [RelatedLinks]  
   keyword [195](#)

RelatedHead, [RelatedLinksText] keyword [194](#)  
 [RelatedLinks], append links to topics [191](#)  
 [RelatedLinksText], labels for related links [194](#)  
 RemoveAHrefAttrs, [HTMLOptions] XML  
   keyword [454](#)  
 RemoveANames, [HTMLOptions] keyword [550](#)  
   for XML link anchors [454](#)  
 RemoveATags, [HTMLOptions] XML keyword [454](#)  
 RemoveChmFilePaths, [MSHtmlHelpOptions]  
   keyword [349](#)  
 RemoveEmptyTableParagraphs, [Table] keyword  
   for DITA [481](#)  
   for DocBook [503](#)  
   for HTML [640](#)  
 RemoveFilePaths, [HTMLOptions] keyword [551](#)  
   identify links to other files [359](#), [553](#)  
 RemoveInternalAnchors, [JavaHelpOptions]  
   keyword [396](#), [409](#)  
 RemoveInternalAnchors, [OracleHelpOptions]  
   keyword [396](#), [409](#)  
 RemoveXrefHotspots, [HTMLOptions]  
   keyword [454](#), [502](#)  
 RepeatMax, [Macros] keyword [706](#)  
 [Required], include unused formats [120](#)  
 ResetAbbrevAt, [Glossary] keyword [205](#)  
 Right, [HTMLParaStyles] format property [573](#)  
 RowAttribute, [HTMLParaStyles] format  
   property [634](#)  
 RowGroup, [HTMLParaStyles] WAI format  
   property [662](#)  
   in header cells to define row groups [661](#)  
   in [Table]RowGroupHead cells [670](#)  
   use with scope method [672](#)  
 RowGroupHead, [Table] WAI keyword [670](#)  
   id/header table cell attribute [670](#)  
   row-group extent [671](#)  
 RowGroupIDs, [Table] WAI keyword [671](#)  
   id/header table cell attribute [670](#)  
   override for selected tables [676](#)  
   row-group extent [671](#)  
   with id/headers method [661](#)  
 RowHead, [Table] WAI keyword [674](#)  
   id/header table cell attribute [670](#)  
   row extent [674](#)  
 RowIDs, [Table] WAI keyword [674](#)

id/header table cell attribute [670](#)  
 override for selected tables [676](#)  
 row extent [674](#)

RowSpanHead, [Table] WAI keyword [673](#)  
 id/header table cell attribute [670](#)  
 row-span extent [673](#)

RowSpanIDs, [Table] WAI keyword [672](#)  
 dependencies [659](#)  
 id/header table cell attribute [670](#)  
 override for selected tables [676](#)  
 row-span extent [673](#)

[RuninHeadText], text of run-in headings [98](#), [154](#)

## S

Scope

[HTMLParaStyles] WAI format property [662](#)  
 [Table]AccessMethod property [658](#)  
   apply scope method to all tables [658](#)  
   avoid redundant attributes [447](#)  
   dependencies [668](#), [677](#)  
   enable [Table\*Attributes] [644](#)  
 [TableAccess] property [659](#)  
   enable [Table\*Attributes] [644](#)

Scope, [Templates] keyword [734](#), [741](#)

ScopeCol, [Table] WAI keyword [668](#)  
 dependencies [659](#)  
 override for selected tables [676](#)

ScopeColGroup, [Table] WAI keyword [668](#)  
 dependencies [659](#)  
 override for selected tables [676](#)

ScopeRow, [Table] WAI keyword [668](#)  
 dependencies [659](#)  
 override for selected tables [676](#)

ScopeRowGroup

[Table] WAI keyword  
   dependencies [659](#)  
   enable [Table\*Attributes] [644](#)  
   override for selected tables [676](#)  
 [TableAccess] override, enable  
   [Table\*Attributes] [644](#)  
 [Tables] WAI keyword [668](#)

script, [ParaTags] format property [566](#)

ScriptType, [HTMLOptions] keyword [568](#)

SearchHighlightStyle, [OmniHelpOptions]  
 keyword [374](#)

SearchWordMin, [OmniHelpOptions]

keyword [373](#)

Sec\*, [JavaHelpOptions] secondary-window  
 properties

SecFontColor [406](#)  
 SecFontFamily [406](#)  
 SecFontSize [406](#)  
 SecFontStyle [406](#)  
 SecFontWeight [406](#)  
 SecGraphic [406](#)  
 SecLocation [405](#)  
 SecName [405](#)  
 SecSize [405](#)  
 SecText [406](#)  
 SecType [406](#)

SecMarkerPrefix, [JavaHelpOptions]  
 keyword [407](#)

[SecWindows], secondary windows [263](#)  
   for HTML Help [333](#)  
   for OmniHelp [371](#)  
   for Oracle Help [408](#)  
   subject to configuration overrides [772](#)

SeeAlsoEnd, [IndexSeeText] keyword [209](#)

[SeeAlsoEndIndex], index see/see-also format  
 building blocks [208](#)

SeeAlsoStartIndex, [IndexSeeText]  
 keyword [209](#)

[SeeAlsoStartIndex], index see/see-also format  
 building blocks [208](#)

SeeAlsoTerm, [Index] keyword [254](#)

SeeEndIndex, [IndexSeeText] keyword [209](#)

[SeeEndIndex], index see/see-also format building  
 blocks [208](#)

SeeStartIndex, [IndexSeeText] keyword [209](#)

[SeeStartIndex], index see/see-also format build-  
 ing blocks [208](#)

SeeTerm, [Index] keyword [254](#)

SelectorIncludesTag, [CSS] keyword [605](#)

[ServiceMark], default service-mark format [157](#)

[Setup]

  options determined at run time, *listed* [733](#)  
 subject to configuration overrides [770](#)

[Setup], set up DITA2Go options [33](#), [69](#)

ShipPath, [Automation] keyword [806](#)

  activated by CompileHelp or FTSCCommand [803](#)

  activated by WrapAndShip [788](#)

ShortdescFormat, [RelatedLinks] keyword [194](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- ShowElementPath, [ElementOptions]  
keyword 72
- ShowLog, [Logging] keyword 74
- ShowNavLeft, [OmniHelpOptions] keyword 364
- ShowSubjects, [OmniHelpOptions] keyword 370
- ShowUndefinedFormats, [Logging] keyword 75
- SiblingHead, [RelatedLinksText] keyword 194
- SimpleTableFormat, [TableOptions]  
keyword 103
- Size, [Base] keyword 577
- SizeN, [HTMLParaStyles] format property 573  
overrides [FontSizes] 577
- SmartSplit, [HTMLOptions] keyword 527
- SortSeeAlsoFirst, [Index] keyword 255
- SpacelessMatch, [Options] keyword 63, 73
- [Spacer], indent images and tables 685
- SpacerAlt, [HTMLOptions] keyword 618
- Spacing, [Table] keyword 636  
overridden by [TableAttributes] 633, 637
- Span, [HTMLParaStyles] WAI format  
property 662  
identify rows and columns 673
- Split  
[HTMLParaStyles] format property 526  
trail dependencies 556  
[MarkerTypes] property 724
- Split\*, [Inserts] split-file keywords:  
SplitBottom 536  
position trails of links 559  
SplitEnd 536  
SplitFrames, 536  
SplitHead 536  
SplitHeadEnd 536  
SplitTop 536
- SplitTopicFiles, [Chunking] keyword 523
- SplitTrail, [Trails] keyword 558  
dependencies 556
- StartingFSButton, [NavigationMacros]  
keyword 563
- StartingPrevFSButton, [NavigationMacros]  
keyword 562
- StartingPrevFSMacro, [NavigationMacros]  
keyword 561
- [StepAttrPrefixes], attributes to determine which  
Step format to use 96
- [StepImportancePrefixes], Step format-name  
prefixes based on importance value 96
- StepsHeadFormat, [ElementOptions]  
keyword 88
- [StopWords], for OmniHelp search 373
- Strike, [HTMLCharStyles] or  
[HTMLCharStyles] format property 573
- StripGraphPath, [Graphics] keyword 70, 611,  
747  
synchronize graphics settings 799  
use referenced graphics without converting 612  
use system commands to manage files 347
- StripTable, [Table] keyword 647
- [Style\*] sections  
*all subject to configuration overrides 773*  
[StyleCellAbbr] 663  
[StyleCellAttribute] 663, 635  
[StyleCellAxis] 663  
[StyleCellScope] 663  
[StyleCodeStore] 693  
[StyleFilePrefix] 784  
[StyleFileSuffix] 784  
[StyleMetaName] 436  
[StyleParaLinkClass] 547  
[StyleRowAttribute] 634  
[StyleTextStore] 693  
[StyleTitlePrefix] 532  
[StyleTitleSuffix] 532  
[StyleTrailPrefix] 556  
[StyleTrailSuffix] 556  
[StyleWindow] 334
- SubFormats, [Templates] keyword 110, 141
- Subformats, [Templates] keyword 730
- Suffix. [\*BList] keyword 215
- Summary, [HTMLParaStyles] WAI table  
keyword 654
- SystemCommandWindow, [Automation]  
keyword 778
- SystemEndCommand, [Automation] keyword 777
- SystemID, [MapGeneration] keyword 70
- SystemStartCommand, [Automation]  
keyword 777
- SystemWrapCommand, [Automation] keyword 777



## T

- [TableAccess], override properties [630, 659, 676](#)
  - override [Table] default access method [659](#)
  - subject to configuration overrides [774](#)
- [TableAfterMacros] [642, 643](#)
  - subject to configuration overrides [774](#)
- [TableAnchorFormats], override anchor paragraph format for selected tables [103](#)
- TableAttributes, [Table] keyword [453, 636, 638](#)
- [TableAttributes]
  - overrides [Attributes] values [633](#)
  - overrides border, cellpadding, and cellspacing in [Attributes] [633](#)
  - overrides [Table]Border, Padding, and Spacing [633](#)
  - subject to configuration overrides [774](#)
  - summary and title [654](#)
- [TableBeforeMacros] [642](#)
  - add space before tables [643](#)
  - invoke macros around tables [642](#)
  - subject to configuration overrides [774](#)
- TableBody, [HTMLParaStyles] WAI table-cell property [664](#)
- [TableBodyAttributes] [644](#)
  - subject to configuration overrides [774](#)
- TableCaptionTag, [Table] keyword [503, 641](#)
- [TableCellAttributes] [645](#)
  - base CSS class on table format [635](#)
  - subject to configuration overrides [774](#)
- [TableCellEndMacros] [645](#)
  - subject to configuration overrides [774](#)
- [TableCellStartMacros] [645](#)
  - selectively modify table text [646](#)
  - subject to configuration overrides [774](#)
- [TableClasses], map table formats to CSS classes [603, 633](#)
- [TableEndMacros]
  - capture row and column counts [645](#)
  - invoke macros around tables [642](#)
  - subject to configuration overrides [774](#)
- [TableFooterAttributes] [644](#)
  - subject to configuration overrides [774](#)
- TableFooterClass, [TableOptions]
  - keyword [104](#)
- TableFooterRows, [Table] keyword [631](#)
  - overridden by [TableAccess] method [631](#)
- TableFootnoteSeparator, [Table] keyword [642](#)
- TableFootnotesWithTable, [Table]
  - keyword [642](#)
- [TableGroup] [626](#)
  - assign with \*Config marker [775](#)
  - subject to configuration overrides [774](#)
- TableHead, [HTMLParaStyles] WAI table-cell property [664](#)
- [TableHeaderAttributes] [644](#)
  - subject to configuration overrides [774](#)
- TableHeaderCols, [Table] keyword [631](#)
  - effect on ScopeRow [668](#)
  - overridden by [TableAccess] method [631](#)
- TableHeaderRows, [Table] keyword [631](#)
  - effect on ScopeCol [668](#)
  - overridden by [TableAccess] method [631](#)
- TableIndents, [Table] keyword [641](#)
- [TableIndents] [641](#)
- [TableOptions], assign format names to tables [103](#)
- [TableOutclassMaps], assign format names to tables [90](#)
- [TableReplaceMacros] [642](#)
  - subject to configuration overrides [774](#)
- [TableRowAttributes] [644](#)
  - subject to configuration overrides [774](#)
- [TableRowEndMacros] [645](#)
  - subject to configuration overrides [774](#)
- [TableRowStartMacros] [645](#)
  - selectively modify table text [646](#)
  - subject to configuration overrides [774](#)
- Tables, [Templates] keyword [110, 129, 730](#)
- [Tables]
  - access method [657–666, 668, 672, 677](#)
  - caption [503, 641](#)
  - cell access method [667–676](#)
  - eliminate attributes for XML [453, 636, 638](#)
  - overridden by [TableAttributes] [633](#)
  - properties [636](#)
  - split-file titles [532](#)
  - structure [627–632](#)
  - subject to configuration overrides [771](#)
- TableSizing, [Table] keyword [638](#)
  - overridden by [TableSizing] [639](#)
- [TableSizing] [639](#)
  - subject to configuration overrides [774](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- [TableStartMacros] [642](#)
  - capture row and column counts [645](#)
  - override column or row groups [630](#)
  - selectively modify table text [646](#)
  - specify <col> elements [629](#)
  - subject to configuration overrides [774](#)
- TableTitle, [HTMLParaStyles] format property [654](#)
- TableTitles, [Table] keyword [641](#)
- [TargetFiles] [444](#)
  - for jumps from image maps [624](#)
  - for jumps to a window type [551](#)
- [Targets] [444](#)
  - for jumps to a window type [551](#)
  - subject to configuration overrides [773](#)
- TasksHead, [RelatedLinksText] keyword [194](#)
- TbFootClass, [CSS] keyword [603](#)
- TblFootFormat, [Footnotes] keyword [102](#)
- TblFootnoteStream, [NumberStreams] keyword [147](#)
- [Templates] [737](#)
  - for document-specific settings [732](#)
  - for general configuration settings [731](#)
  - for macro libraries [729](#)
  - format configuration [110](#)
- Text, [XrefStyles] format property [552](#)
- TextStore, [HTMLParaStyles] format property [693](#)
  - create a macro variable [688](#)
- Title
  - [HTMLOptions] keyword [435](#), [533](#)
  - [HTMLParaStyles] format property [436](#)
    - for split files [532](#)
    - trail dependencies [556](#)
  - [JavaHelp window] parameter [404](#)
  - [MarkerTypes] property [724](#)
- TitleFormat. [\*BList] keyword [215](#)
- TitleOnlyTopicID, [TopicHeads] keyword [106](#)
- TitleOnlyTopicType, [TopicHeads] keyword [106](#)
- [Titles]
  - for individual output files [436](#)
  - overrides [HTMLParaStyles]Title [533](#)
  - precedence [531](#)
- Toc\*
  - [EclipseHelpOptions] keywords:
    - TocExtradir [419](#)
    - TocFilename [419](#)
    - TocLabel [421](#)
    - TocLinkTo [424](#)
    - TocNamesFileOnly [422](#)
    - TocPrimary, [419](#)
    - TocTopic [421](#)
  - [JavaHelpOptions] keywords:
    - TocClosedImage [396](#)
    - TocOpenImage [396](#)
    - TocTopicImage [396](#)
  - [OmniHelpOptions] keywords:
    - TocButtons [369](#)
    - TocExpand [368](#)
    - TocGroupsOpen [369](#)
    - TocIcoBase [369](#)
    - TocOpenLevel [369](#)
- TOCFile, [Contents] keyword [199](#)
- TOCFormat. [\*BList] keyword [216](#)
- TOCFormat, [Contents] keyword [200](#)
- TocIdxFilePrefix, [EclipseHelpOptions] keyword [421](#)
- [TocLevelExpand], JavaHelp 2 settings [396](#)
- [TocLevelImage], JavaHelp 2 settings [396](#)
- TOCSuffix, [Contents] keyword [199](#)
- TOCTitle, [ContentsText] keyword [191](#), [199](#)
- TOCTitleFormat, [Contents] keyword [200](#)
- TOCXrefFormat, [Contents] keyword [200](#)
- Toolbar, [JavaHelp window] parameter [404](#)
- Top
  - [Inserts] keyword [535](#), [536](#), [710](#)
    - to position a navigation macro [564](#)
    - to position trails of links [558](#)
  - [JavaHelp window] parameter [404](#)
- TopButton, [NavigationMacros] keyword [563](#)
- TopFirst, [OmniHelpOptions] keyword [363](#)
- TopHigh, [OmniHelpOptions] keyword [363](#)
- TopicBreak, [Inserts] keyword [526](#), [535](#), [710](#)
- TopicHeadChildHeadFormat, [TopicHeads] keyword [107](#)
- [TopicHeads] [200](#)
- [TopicHeads], configure map output [106](#)
- TopicheadsHaveNavtitles, [TopicHeads] keyword [200](#)
- [TopicHeadText], heading for list of child



topics [107](#)  
 TopicStartCode  
     [MarkerTypes] property [724](#)  
 TopicTitleFormat, [RelatedLinks]  
     keyword [194](#)  
 TopMacro, [NavigationMacros] keyword [561](#)  
 [TradeMark], default trademark format [157](#)  
 Trail, [HTMLParaStyles] format property [556](#)  
     dependencies [556](#)  
     for non-heading formats [558](#)  
     required for first paragraph in file [559](#)  
 Trail\*, [Trails] keywords:  
     TrailCurrent [557](#)  
     TrailEnd [557](#)  
     TrailIndent [557](#)  
     TrailPosition [558](#)  
     TrailSep [557](#)  
     TrailStart [557](#)  
 [TrailLevels] [558](#)  
     subject to configuration overrides [773](#)  
 [Trails], bread-crumbs link list [556–558](#)  
     subject to configuration overrides [771](#)  
 TreatTableFigAsTable, [FigureOptions]  
     keyword [203](#)  
 TreatTopicheadsAsTopics, [TopicHeads]  
     keyword [106](#)  
 [Typographics] [466, 579](#)

## U

ULine, [HTMLCharStyles] or [HTMLCharStyles]  
     format property [573](#)  
 UnicodeFTS  
     [OmniHelpOptions] keyword [373](#)  
 UnicodeLocale  
     [OmniHelpOptions] keyword [373](#)  
 UniqueNameSuffixFormat, [IDOptions]  
     keyword [77](#)  
 UniqueNameSuffixLength, [IDOptions]  
     keyword [77](#)  
 [UnorderedListElement], implied list format [126](#)  
 UnwrapPRE, [HTMLOptions] keyword [439, 581](#)  
 URLTarget, [HTMLOptions] keyword [554](#)  
 UseAbbrevInTitles, [Glossary] keyword [206](#)  
 UseAddedDivider, [RelatedLinks] keyword [195](#)  
 UseAliasAName, [MSHtmlHelpOptions]  
     keyword [341](#)  
 UseAllInTOC, [MapOptions] keyword [201](#)  
 UseAncestors, [RelatedLinks] keyword [190](#)  
 UseAnums, [HTMLOptions] keyword  
     for HTML output [568](#)  
     for XML output [453](#)  
     XML default value [449](#)  
 UseBackForward, [OmniHelpOptions]  
     keyword [364](#)  
 UseBranchKeydefs, KeyOptions keyword [172](#)  
 UseCALSTable, [Table] keyword [503, 627](#)  
     XML default value [449](#)  
 UseCatalogs, [Catalogs] keyword [67](#)  
 UseCharacterTypographics, [Typographics]  
     keyword [580](#)  
 UseCharRangeClasses, [CSS] keyword [603](#)  
 UseChildren, [RelatedLinks] keyword [190](#)  
 UseChmInLinks, [MSHtmlHelpOptions]  
     keyword [324](#)  
 UseCodePage  
     [MSHtmlHelpOptions] keyword [317](#)  
 UseCommaAsSeparator, [Index] keyword [253](#)  
 UseCompactForm, [Index] keyword [209](#)  
 UseCompositeDropJS, [DropDowns] keyword [269](#)  
 UseConditionalFlagging, ConditionOptions  
     keyword [165](#)  
 UseContext, [EclipseHelpOptions]  
     keyword [420](#)  
 UseCousins, [RelatedLinks] keyword [190](#)  
 UseCSS, [CSS] keyword [593](#)  
     affects default value of  
         [HTMLOptions]AlignAttributes [573](#)  
         [HTMLOptions]Basefont [577](#)  
         LinkClassIsParaClass [547](#)  
         NoClassLists [588](#)  
     affects default values of other settings [597](#)  
     affects use of <font> tags [609](#)  
     replaces [HtmlOptions]Stylesheet [596](#)  
 UseCSSLeading, [HTMLOptions] keyword [609](#)  
 UseDCLOutput, [Automation] keyword [44](#)  
 UseDCLSource, [Automation] keyword [43](#)  
 UseDefaultStopWords, [OmniHelpOptions]  
     keyword [374](#)  
 UseDescendants, [RelatedLinks] keyword [190](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- UseDOCTYPE, [HTMLOptions] keyword [432](#), [502](#)
- UseDropDowns, [DropDowns] keyword [265](#)
- UseElementNameForFormat, [ElementOptions]  
keyword [88](#)
- UseExistingDCL, [Setup] keyword [73](#)
- UseFavorites, [JavaHelpOptions] keyword [393](#)
- UseFigureAnchor, [FigureOptions]  
keyword [103](#)
- UseFontFace, [HTMLOptions] keyword [577](#), [578](#)
- UseFontSize, [HTMLOptions] keyword [579](#)
- UseFootnoteLists, [HTMLOptions] keyword [583](#)
- UseFootXrefTag, [HTMLOptions] keyword [503](#),  
[582](#)  
XML default value [449](#)
- UseFormatTypographics, [Typographics]  
keyword [580](#)
- UseFrameSet, [HTMLOptions] keyword [444](#)
- UseFTS  
[EclipseHelpOptions] keyword [420](#)  
[JavaHelpOptions] keyword [397](#)  
[MSHtmlHelpOptions] keyword [340](#)  
[OmniHelpOptions] keyword [372](#)
- UseFullPath, [Options] keyword [70](#)
- UseGlossary, [JavaHelpOptions] keyword [402](#)
- UseHash, [HTMLOptions] keyword [454](#), [502](#)
- UseHeadAndBody, [HTMLOptions] keyword [437](#),  
[451](#)  
XML default value [449](#)
- UseHeading, [\*BList] keyword [215](#)
- UseHideShow, [OmniHelpOptions] keyword [364](#)
- UseHVIndex, [Index] keyword [252](#)
- UseIndex, [EclipseHelpOptions] keyword [419](#)
- UseIndexentryTag, [JavaHelpOptions]  
keyword [397](#)
- UseIndexentryTag, [OracleHelpOptions]  
keyword [397](#)
- UseIndexHeading, [Index] keyword [207](#)
- UseIndexLetters, [Index] keyword [210](#)
- UseIndexLevelNum, [Index] keyword [212](#)
- UseIndexTopLetters, [Index] keyword [210](#)
- UseInformaltableTag, [Table] keyword [503](#),  
[641](#)
- UseLetters, [\*BList] keyword [216](#)
- UseListButton, [OmniHelpOptions]  
keyword [364](#)
- UseListedXrefFilesOnly, [HTMLOptions]  
keyword [454](#), [502](#)
- UseListTypeAttribute, [CSS] keyword [395](#), [588](#)
- UseLog, [Logging] keyword [74](#)
- UseManifest, [EclipseHelpOptions]  
keyword [416](#)
- UseMapDescAsTitle, [MapOptions] keyword [106](#)
- UseNavButtons, [NavigationMacros]  
keyword [560](#)
- UseNavtitleMarkers  
[JavaHelpOptions] keyword [250](#)  
[OmniHelpOptions] keyword [250](#)  
[OracleHelpOptions] keyword [250](#)
- UseNestedTopicsInTOC, [Contents]  
keyword [199](#)
- UseOutputClassForFormat, [ElementOptions]  
keyword [88](#), [145](#)
- UseParagraphTypographics, [Typographics]  
keyword [580](#)
- UseParent, [RelatedLinks] keyword [190](#)
- UsePlugin, [EclipseHelpOptions] keyword [415](#)
- UsePrevNext, [OmniHelpOptions] keyword [364](#)
- UsePrevNext, [RelatedLinks] keyword [190](#)
- UsePxSuffix, [Graphics] keyword [622](#)
- UseRawName, [HTMLOptions] keyword [783](#)
- UseRawNewlinks, [HTMLOptions] keyword [279](#),  
[410](#)
- UseRelatedDivider, [RelatedLinks]  
keyword [195](#)
- UseRelDescAsTitle, [RelatedLinks]  
keyword [192](#)
- UseRelDescription, [RelatedLinks]  
keyword [191](#)
- UseRuntime  
[EclipseHelpOptions] keyword [420](#)
- UseSearchHighlight, [OmniHelpOptions]  
keyword [374](#)
- UseSiblings, [RelatedLinks] keyword [190](#)
- UseSingleton, [EclipseHelpOptions]  
keyword [417](#)
- UseSpacers, [HTMLOptions] keyword [618](#), [641](#)
- UseSpanAsDefault, [CSS] keyword [602](#)
- UseStart, [OmniHelpOptions] keyword [364](#)

UseSubHelpSets, [JavaHelpOptions]  
     keyword 410  
 UseSubHelpSets, [OracleHelpOptions]  
     keyword 410  
 UseTableAnchor, [TableOptions] keyword 103  
 UseTbFootnoteLists, [HTMLOptions]  
     keyword 583  
 UseTbHeaderCode, [Table] keyword 628  
 UseTitleForAlt, [Graphics] keyword 623  
 UseTOCDescriptions, [Contents] keyword 200  
 UseTopButtons, [OmniHelpOptions]  
     keyword 364  
 UseTopicShortdesc, [MapGeneration]  
     keyword 69  
 UseTopicTypes, [RelatedLinks] keyword 191  
 UseTopLetters, [\*BList] keyword 216  
 UseTypographicElements, [Typographics]  
     keyword 466, 579  
 UseTypographicStyles, [Typographics]  
     keyword 580  
 UseUlink, [HTMLOptions] keyword 454, 502  
 UseXMLDeclaration, [HTMLOptions]  
     keyword 438  
 UseXMLRoot, [HTMLOptions] keyword 437, 502

## V

ValidOnly, [HTMLOptions] keyword 445  
 [VariableMaps], assign variable names to  
     elements 186  
 ViewOutputCommand, [HTMLOptions]  
     keyword 443  
 ViewOutputCommand, [\*Options] keyword 35,  
     248  
 ViewOutputFile, [HTMLOptions] keyword 443

## W

WhileMax, [Macros] keyword 705  
 Width, [JavaHelp window] parameter 404  
 WildcardMatch, [Options] keyword 73  
 Window  
     [MarkerTypes] property 724  
 Window, [HTMLParaStyles] format property 333

Windows, [JavaHelpOptions] keyword  
     list of windows 403  
 WrapAndShip, [Automation] keyword 788  
 WrapCopyFiles, [Automation] keyword 793  
     activated by CompileHelp or FTSCCommand 803  
 WrapPath, [Automation] keyword 792  
     activated by CompileHelp or FTSCCommand 803  
     activated by WrapAndShip 788  
     for JavaHelp, Oracle Help 389, 392  
 WriteClassAttributes, [CSS] keyword 593  
     default depends on UseCSS 597  
     replaces [HtmlOptions] Stylesheet 596  
     turn off for XML 452, 604  
 WriteContext, [EclipseHelpOptions]  
     keyword 420  
 WriteCssLink, [CSS] keyword 593  
     change CSS mid-document 598  
     customize CSS link tag 599  
     default depends on UseCSS 597  
     replaces [HtmlOptions] Stylesheet 596  
     select CSS file at run time 597  
     use with CssFileName 595  
 WriteCssStylesheet, [CSS] keyword 593  
     default depends on UseCSS 597  
     designate CSS file 595  
     replaces [HtmlOptions] Stylesheet 596  
 WriteDropIconFiles, [DropDowns] keyword 269  
 WriteDropJSFile, [DropDowns] keyword 273  
 WriteFlagsFile, ConditionOptions  
     keyword 165  
 WriteHelpProjectFile, [MSHtmlHelpOptions]  
     keyword 318  
 WriteHelpSetFile, [JavaHelpOptions]  
     keyword 393  
 WriteHelpSetFile, [OracleHelpOptions]  
     keyword 393  
 WriteIndexCssLink, [Index] keyword 211  
 WriteManifest, [EclipseHelpOptions]  
     keyword 417  
 WritePlugin, [EclipseHelpOptions]  
     keyword 418  
 WriteSpacerFile, [HTMLOptions] keyword 618

## X

XHLangAttr, [HTMLOptions] keyword 433, 502

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

XHLanguage, [HTMLOptions] keyword [433](#)  
 XHNamespace, [HTMLOptions] keyword [433](#)  
 XMLEncoding, [HTMLOptions] keyword [450](#)  
     for double-byte characters [434](#)  
 XMLLinkAttrs, [HTMLOptions] keyword [454](#)  
 XMLRoot, [HTMLOptions] keyword [437](#), [451](#)  
     XML default value [449](#)  
 XMLVersion, [HTMLOptions] keyword [450](#)  
 [XrefFiles], interfile links [554](#)  
 XrefFormat. [\*BList] keyword [216](#)  
 XrefFormatIsXrefClass, [CSS] keyword [604](#)  
     default depends on UseCSS [597](#)  
 XrefFtnFormat, [ElementOptions] keyword [101](#)  
 XrefNumFormat, [ElementOptions] keyword [101](#)  
 [XrefOutclassMaps], format names for cross  
     references [91](#)  
 XrefSpaceChar, [HTMLOptions] keyword [549](#)  
 [XrefStyleLinkSrc], macro for href  
     attribute [552](#)  
     for KeyHelp pop-ups [323](#)  
     subject to configuration overrides [773](#)  
 [XrefStyles], cross-reference format [552](#)  
     for KeyHelp pop-ups [323](#)  
     subject to configuration overrides [773](#)  
 XrefTextFormat, [ElementOptions]  
     keyword [101](#)  
 XrefTitleFormat, [ElementOptions]  
     keyword [101](#)

## Z

ZeroCSSMargins, [CSS] keyword [452](#)  
 ZipCommand, [EclipseHelpOptions]  
     keyword [427](#)  
 ZipParams, [EclipseHelpOptions] keyword [428](#)

### A

- `<a name=...>` tags, suppressing line breaks in 439
- abbr, HTML table attribute for WAI
  - guidelines for using 653
  - via `CellAbbr` marker 666
  - via `[HtmlStyleCellAbbr]` 663
  - via paragraph format 662
  - via special paragraph format 666
- abbreviations of glossary terms 205
  - class attribute for `<abbreviated-form>`, specifying 206
  - first-use rules, overriding 206
  - first-use rules, specifying 205
  - formatting for output 206
- absolute vs. relative paths
  - in configuration settings 64
  - in graphics references 612
- accented characters, converting to HTML 574
- ActiveX and MS HTML Help 245
- adaptive table sizing
  - for HTML 639
  - overriding 639
  - for WinHelp 291
- `<address>`, HTML paragraph tag 566
- after, macro string operator 707
- after, output format property 118
- alert** markers
  - for HTML Help pop-ups 264
  - for HTML split points 527
  - for WinHelp pop-ups 301
- alert** pop-ups, creating for WinHelp 301
- alerttitle** markers
  - for WinHelp pop-ups 301
- alias files for context-sensitive Help 278
- align, HTML attribute
  - and `valign`, automatically generated, excluding from table cells 453, 635
  - eliminating from paragraph tags 573
  - for HTML Help contents entries 336
- aligning
  - graphics for HTML 617
- ALink
  - See also* ALinks
  - jumps, configuring for HTML-based Help 261
  - keywords
    - adding with format properties 260
    - adding with markers 260
  - list destinations, specifying 262
- ALink**, PI marker for Help systems 719
- ALinks
  - See also* ALink
  - DITA, in relationship tables 496
  - generating from related links for Help output 192
  - HTML Help
    - creating 325
    - target-and-jump 329
    - uncompiled 326
  - OmniHelp, support for 370
  - Oracle Help for Java, creating 409
  - target-and-jump, for HTML-based Help 262
  - understanding 259
  - WinHelp
    - adding footnotes to topics 304
    - configuring 303
- alt, HTML attribute
  - empty, omitting 620
  - for drop-down icons 269
  - for image maps 622
  - for images
    - via **GraphAlt** marker 651
    - via graphic file name 620
    - via special format 651
    - WAI guidelines for using 650
  - for links, via special format 652
- Altura
  - graphics format for WinHelp 293
  - QuickHelp, specifying for WinHelp 283
- anchor tags, suppressing line breaks in 439
- anchors, internal, for JavaHelp and Oracle Help 409
- Anum, `[HtmlStyles]` format property, retain autonumbers 453
- archiving files for delivery 802
- arithmetic operators for macro expressions, *listed* 702
- arrays

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- initializing 696
  - instead of conditional expressions 698
  - processing with macros 696
  - processing with pointers 697
  - ASCII
    - decimal character code, mapping for HTML 574
    - text output via conversion to Word 241
  - Asian languages, HTML Help support for 344
  - aspect ratio, preserving for graphics, HTML 621
  - assembling files for distribution 792
    - for Eclipse Help 427
    - for JavaHelp and Oracle Help 389
    - for OmniHelp 380
  - assembly directory
    - default files copied to 795
    - emptying before copying to 793
    - files to copy, specifying 793
    - graphics files to copy, listing 797
    - specifying 792
  - associative links, *see* [ALinks](#)
  - attribute markers for HTML or XML 721
    - listed 722
    - names of 721
  - attribute values
    - defaults for missing 100
    - including or excluding content based on 164
    - passing through to output 165
    - prefixing format names with 95
  - attributes, DITA, *see* [DITA, attributes](#)
  - attributes, DocBook, *see* [DocBook, attributes](#)
  - attributes, HTML
    - align, omitting from paragraph tags 573
    - assigning
      - to character formats 569
      - to paragraph formats 566
    - image size, omitting 620
    - image, specifying 619
    - link class, assigning with a marker 546
    - link, assigning with markers 547
    - table, assigning with markers 634
    - table, automatically generated, eliminating 453, 635
    - WAI, assigning values to 663
  - attributes, image, *see* [graphics, attributes](#)
  - automating
    - conversions 765, 777
    - production of deliverables 787
  - autonumbers
    - converting for database input 780
    - converting to HTML 567
    - converting to Word 226
    - including or excluding
      - for generic XML 453
      - for HTML 567
    - tabs in, eliminating for HTML 568
  - axis, HTML table attribute for WAI
    - guidelines for using 653
    - identifying cells by virtual properties 669
    - via `Axis` format property 662
    - via `AxisVal` format property 666
    - via **CellAxis** marker 666
    - via `[HtmlStyleCellAxis]` property 663
- ## B
- background image, for HTML 624
  - backslash
    - character literal for macro variables 689
    - escape character
      - in frame code for OmniHelp 364
      - in KLink jumps 261
      - in macros 680
      - in RTF code 238
    - prohibited in catalog paths 68
    - separator in file paths
      - for HTML Help 349
      - in code markers for HTML 441
    - trailing, to remove line breaks in macros 681, 781
  - base values of format properties, specifying 120
  - based, output format property 116
    - vs. `inline` 122
  - `$_basefile`, macro variable 537, 691
  - `$_basename`, system-command variable 691, 778
  - `$_basetitle`, macro variable 537, 691
  - .bat file for system commands 779
  - before, macro string operator 707
  - before, output format property 118
  - beta executables 37
  - beta version of **DITA2Go**, running via DCL 46
  - Bezier curves, in WMF graphics 746
  - `bgcolor`, automatically generated, excluding from HTML table cells 453, 635



- binary TOC for HTML Help 321
  - for browse buttons 320, 335
  - mid-topic links 337
  - no-link contents entries 337
- bitmaps
  - compressing 235
  - resolution of, for RTF 235
- bitwise operators for macro expressions, *listed* 701
- blank paragraphs, *see* [empty paragraphs](#)
- blanks, *see* [spaces](#)
- block output formats
  - border properties of 128
  - properties of, *listed* 124
- block text, content-model element type 760
- <blockquote>, HTML paragraph tag 566
- blocks for expandable sections
  - configuring 271
  - delimiting
    - with formats 266
    - with markers 267
- blue borders, eliminating from images 619
- BMP, graphics export format
  - See also* [bitmaps](#)
- BMROOT entries in .hpx files 286
- bold, as a format override for HTML 573
- book-level maps, DITA, vs. chapter maps 494
- bookmarks, Word
  - for cross references 229
  - limiting 221
- border formats, defining 144
- border properties
  - for RTF output pages 137
  - of block formats 128
- border, automatically generated, excluding from
  - HTML tables 453, 635
- BorderFormat**, PI marker for specifying border
  - subformats 719
- borders
  - around images, hiding 619
  - around table cells, HTML 636, 637
  - for output formats
    - around RTF pages 137
    - around RTF sections 135
    - around table cells 133
    - around table rows 132
    - around tables 131
  - around text 128
  - from <outputclass> attributes 88
- BorderStyle**, PI marker for specifying border
  - subformats 146, 719
- branch** PI marker for scoping in maps 170
- Branch**, PI marker for naming map branches 719
- branches, named, in maps 170
- branching browse schemes for WinHelp 311
- breadcrumb trails
  - in Eclipse Help 415, 422
  - in HTML 555
- <\$\_break>, control structure for macros 704, 706
- breaks, column, inserting in DITA via PIs 137
- breaks, line
  - suppressing
    - in HTML or XML output 439
    - in XML output 451
- breaks, page
  - and section
    - for WinHelp 285
- breaks, topic, including space or a separator in 525
- Bristol HyperHelp
  - format for WinHelp graphics 293
  - specifying for WinHelp 283
- browse
  - buttons, enabling in HTML Help Workshop 319
  - numbers, WinHelp, specifying 310
  - prefix, assigning for branches, WinHelp 311
  - sequences
    - HTML, creating 559
    - WinHelp, creating 310
- browser-dependent
  - HTML list styles 586
  - settings for HTML tables 628
- browsers
  - cookies for, created by OmniHelp 365
  - CSS support in 430, 432, 591
  - font rendering differences 579
  - graphics support in 746
- build numbers, finding
  - in output 820
  - on Web site 820
- bulleted lists, converting
  - to DITA XML 460
  - to DocBook XML 504
  - to HTML 584



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

to Word 226

## bullets

- eliminating from HTML output 568
- for HTML list styles 126
- mapping to special characters for HTML 575
- replacing for W3C validity 446
- specifying for bulleted lists 152

buttons for drop-down links, configuring 269

## C

Calabash XInclude 178

Calibre, for ePub 430

## CALS table model

- default for XML 449
- specifying for HTML 627

cancelling a conversion 38

<caption> tags for HTML tables, placing 641

cascading style sheets, *see* CSS

## case sensitivity

- of ALink keywords 258, 260
- of attribute names for HTML links 549
- of command-line arguments 810
- of CSS class names 600, 601
- of file names 782
- of format names, specifying 73
- of index terms for HTML-based Help 257
- of key names in configuration settings 63, 768
- of KLink keywords 258, 261
- of macro operators 702

catalogs, XML, connecting to 67

**Cell\***, custom markers for HTML table cell attributes 719

**CellAbbr**, custom marker for WAI 666

**CellAxis**, custom marker for WAI 666

**CellClass**, custom marker for CSS 635, 719

**CellGroup**, custom marker for WAI 666, 719

- using to define a ColGroup cell 661
- using to define a RowGroup cell 661

**CellID**, custom marker for WAI 666, 719

cellpadding, automatically generated, excluding from HTML tables 453, 635

cells, *see* table cells; tables

**CellScope**, custom marker for WAI 666, 719

cellspacing, automatically generated, excluding

from HTML tables 453, 635

**CellSpan**, custom marker for WAI 666, 673, 719

## chapter

maps, DITA, vs. book maps 494

char, macro string operator 707

## character

*See also* characters

## encoding

- for code pages 576
- for HTML 434
- for HTML Help 314
- for XML 450

entity references, HTML 570

## formats

- converting to RTF 227
- mapping to CSS `span` classes 602
- output properties of, *listed* 123
- output, default, specifying 121, 122
- properties, overriding 771
- replacing with code, for WinHelp 287, 711
- replacing with code, for Word 228, 711

## literals

- assigning to macro variables 689
- for macro variables, *listed* 689

ranges, Unicode, assigning CSS classes to 603

spans, changing properties of 771

## characters

*See also* character

accented, converting to HTML 574

double-byte, in HTML 434

double-byte, in XML 450

## high ASCII

- encoding for HTML 434
- encoding for XML 450
- mapping to HTML 570
- replacing for W3C validation 445

problem, in HTML hypertext links 548

special, avoiding in URIs 576

## special, converting

- for HTML 570

special, mapping 574

- for code pages 576

## Chinese

**DITA2Go** support for 27

for HTML Help output, specifying 345

for RTF output, specifying 220

.chm file, compiled HTML Help 314, 348

CHM files, merging 350

- CHM files, unblocking 314
- chrome, browser, for OmniHelp 363
- chunking DITA maps 523
- chunking policy, specifying 524
- CJK languages 27, 434
- class attribute
  - adding to element 183
  - assigning element type properties to 183
- class, CSS attribute
  - See also* CSS, class names
  - for graphics 615
  - for links 546, 720
    - assigning with a format 547
    - assigning with a marker 546
  - for paragraphs 601
  - for table cells 635
  - for table columns 629
  - for tables 603, 633
  - for Unicode character ranges 603
  - for XML output 452
  - naming restrictions 600
- \$\$\_class, macro variable 691
- closing `</p>` tags, suppressing in HTML 438
- .cnt file, WinHelp contents 306
- code pages
  - encoding for special characters 576
  - for Asian and Cyrillic languages
    - for HTML Help 344
    - for print RTF 220
    - obtaining 28
  - for HTML Help 314
  - omitting, for uncompiled HTML Help 317
- code sections, configuring for HTML 581
- code snippets, external, referencing 175
  - with fragment identifiers 177
  - with processing instructions 177
- Code**, HTML custom marker type 719
- `<col>` element tags for HTML tables 629
- ColGroup and RowGroup cells 676
  - using with id/headers method 670
  - using with scope attributes 668
- `<colgroup>` elements
  - and ColGroup cells 676
  - required for scope column groups 668
  - tags for HTML tables 628
- color number, RTF, assigning via macro 119
- `<$_colnum()>`, predefined macro for RTF output 119, 684
- colors
  - for HTML links 545
  - in graphics
    - for WinHelp 745
    - for Word 745
  - in tables
    - for WinHelp 291
    - for Word 233
  - of bullet symbols, specifying 152
  - text, defining for HTML 439
  - text, specifying
    - for HTML 574, 580
  - text, suppressing, for HTML 573
  - Web-safe
    - for HTML text 440
    - listed* 441
- colspan, HTML table attribute 653
- column breaks, inserting via PIs 137, 138
- column spans in HTML tables 673
- command-line version, *see* DCL, **DITA2Go** command-line version
- commands, system
  - executing 777
  - in batch files 779
  - in **DITA2Go** macros 779
  - understanding 778
- commenting out configuration sections 66
- comments
  - draft, excluding from output 105
  - draft, specifying formats for 105
  - in configuration files, syntax for 64
  - in macro definitions 681
  - in system-command macros 780
  - substituting for paragraphs in HTML 568
- comparison tool, file, obtaining 35
- compiling
  - Help files for delivery 802
  - HTML Help 346
  - JavaHelp with Helen 395
  - WinHelp 284, 285
- complex otherprops values, processing 162
- conditional
  - action settings, syntax of 163
  - actions, defining 162
  - content flagging 163
  - exclusion from output 164

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- expressions
  - in macros 704
  - replacing with list variables 698
- flags
  - See also* flags
  - configuring 166
  - file, CSS 165
  - for assigning attributes 169
  - setting 165
- inclusion in output 164
- operators for macro expressions, *listed* 702
- processing, specifying 161
- conditions
  - default, specifying actions for 163
  - extracting from ditaval files 161
- Config**, custom marker type 719
- configuration
  - file, *see* configuration file
  - macros, *see* configuration macros
  - markers for overrides 767
  - options determined at run time, *listed* 733
  - section, *see* configuration file, sections; configuration sections
  - settings, *see* configuration settings
  - variables, *see* configuration variables
- configuration file
  - comment syntax 64
  - creating
    - for individual ditamap files 765
  - deciding which to edit 734
  - document-specific
    - name of 35
    - where to keep 733
  - editing 49
  - macro, editing 740
  - output-specific, editing 738
  - project, editing 737
  - sections
    - See also* configuration sections
    - names of 63
    - order of 63
  - source-specific, editing 738
  - starting, copied to output directory 39
  - structure of 62
- configuration macros
  - accessing settings with 699
  - changing table settings with 647
  - deploying 700
  - overriding configuration settings with 767
- configuration sections
  - See also* configuration file, sections
  - commenting out 66
  - fixed-key, *listed* 770
  - names of 63
  - order of 63
  - using as list variables 696
  - variable-key
    - cross-reference format, *listed* 773
    - graphic, *listed* 775
    - table format or ID, *listed* 774
    - text format, *listed* 772
- configuration settings
  - case sensitivity of 63, 768
  - changing on the fly 765
  - file-path separator in 64
  - fixed-key
    - overriding 770
    - vs. variable-key 63
  - in configuration templates 742
  - order of 63
  - overrides to, persistent vs. temporary 767
  - overriding 766, 775
    - with macros 767
    - with markers 767
  - precedence of, *listed* 766
  - querying with configuration variables 699
  - rules for 62
  - spaces and tabs in 63
  - syntax of 62
  - variable-key, overriding 771
  - wildcards in 65, 73
- configuration templates
  - See also* templates, configuration 727
  - chaining 743, 765
  - creating 741
  - for **DITA2Map** 816
  - format, setting up 110
  - referencing 731
- configuration variables
  - assigning macros and variables to 768
  - assigning values to 768
  - capturing settings with 699
- Confluence, generating XHTML for 442
- conkeyref
  - cannot be used map to map 175
  - processed along with keyrefs 176
- conref
  - map to map pull 175

- ConrefBranch**, PI marker 171, 719
- Conrefbranch**, PI marker 173
- conrefs
  - in map branches 171
- conrefs in map branches 173
- contains, macro string operator 707
- content model
  - See also* [content models](#); [DTD](#)
  - abstracting from a DTD
    - for DITA 755
    - for DocBook 755
  - configuration file, producing 755
  - configuration sections 756
    - [ElementSets] 758
    - [Topic] 757
    - [TopicFirst] 758
    - [TopicLevels] 759
    - [TopicParents] 758
  - DITA
    - settings, overriding 762
    - topic type, naming 756
  - DocBook, naming 756
  - element levels 759
  - element parents 758
  - element sets 758
  - element types 760
  - first-child elements 758
  - generating from a DTD 754
  - replacing 755, 757
  - root element, specifying 757
- content models
  - See also* [content model](#)
  - built-in
    - configurations for, *listed* 754
    - derivation 753
    - obtaining copies of 754
  - debugging 764
  - preparing for use 755
  - working with 753
- content reference, *see* [conrefs](#)
- content type, XML, specifying 451
- content, adding via format properties 118
- contents
  - Eclipse Help
    - creating 420
    - entries, merging from multiple files 423
    - link paths, supplying 421
    - properties, configuring 421
    - properties, specifying 419
  - HTML Help
    - and index, generating 334
    - entries, configuring 337
    - links to mid-file topics 337
  - HTML-based Help
    - levels, setting 251
  - JavaHelp
    - creating 395
    - entries, configuring 395
    - expansion levels, specifying 396
    - images, designating 396
  - link to, following related links 191
  - mid-topic links, avoiding for Help systems 250
  - OmniHelp, including 367
  - WinHelp
    - assembling for multiple topic files 309
    - combined file, specifying 284
    - configuring 306
    - entries, configuring 306
    - level for headings, specifying 284
    - levels, setting 250
    - multiple files, referencing 309
    - referencing secondary windows 309
  - Word
    - table of, including or excluding 198, 221
- contents, table of (TOC), file name for 199
- context IDs for Eclipse Help 426
- context-sensitive Help 277
  - for Eclipse Help, setting up 425
  - for HTML Help, setting up 340
  - for JavaHelp, using symbolic IDs 410
  - for OmniHelp, setting up 375
- <\$\_continue>, control structure for macros 704, 706
- controls for macro expressions
  - listed* 704
  - using 704
- conversion
  - DCL modules, writing 813
  - events, logging 74
  - project, setting up 39
  - restrictions on file names 26
  - setting up
    - for DITA XML 455
    - for DocBook XML 500
    - for Eclipse Help 413
    - for OmniHelp 357
    - for XML 449

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- settings, *see* configuration settings
- to HTML
  - validating 434
- to WinHelp
  - basic options 284
  - preparing for 282
- via DCL, preparing for 809
- converting to HTML
  - footnotes 581
  - graphics 611
  - list formats 584
  - nested lists 587
  - numbered lists 586
  - special characters 570
  - table footnotes 642
  - tables 625, 648
- converting to RTF
  - character formats 227
  - cross references 229
  - graphics 234
  - tables 232
- converting via DCL
  - single file, example 809
- cookies, OmniHelp, persistence of 365, 381
- copyright statement and date, for WinHelp 286
- copy-to attribute, to override default file names 524
- \$\$\_count, macro variable 692, 706
- cover page
  - break before TOC 525
- crash, debugging 821
- cross references
  - for DITA XML
    - converted to <xref> elements 489
    - format attribute, specifying 491
    - links below topic level 490
    - omitting from footnotes 490
    - scope attribute, specifying 490
    - type attribute, specifying 491
    - understanding how converted 489
    - wrapper outputclass, specifying 489
  - for DocBook XML
    - converted to <xref> elements 507
  - for HTML
    - converting to text 552
    - deleting 552
    - omitting paths from 553
  - for print RTF 229
  - external, enabling 231
  - locking 229
  - omitting from output 231
  - for WinHelp
    - converting 288
    - converting to text 290
    - deleting 290
  - in named map branches 171
- cross-file links, *see* interfile links
- cross-reference
  - branch PI markers 171
  - elements, mapping to output formats 91
  - formats, defining 155
  - formats, specifying 101
  - jump destinations, WinHelp, specifying 289
  - links, in DITA XML 489
  - marker text, truncating in WinHelp 290
  - output format names, default, *listed* 157
  - properties, overriding 773
  - wrapper elements, mapping to output formats 91
- CSH, *see* context-sensitive Help
- CSS 591, 609
  - browsers, supporting 430
  - class attributes for table columns 629, 630
  - class names
    - case of 600
    - for character formats 602
    - for footnotes 603
    - for links, assigning with formats 547
    - for links, assigning with markers 546
    - for paragraph formats 601
    - for table cells 635
    - for table footnotes 603
    - for tables 603, 633
    - for XML tags 452
    - restrictions on 600
  - directory to copy files from, specifying 800
  - file name, specifying 595
  - file, specifying when to create 593
  - files
    - for conditional flags 165
    - for OmniHelp 361
    - list of, to copy 801
  - flags file, specifying 165
  - font sizes, mapping 577
  - font-size units, changing 607
  - for OmniHelp navigation panel, modifying 365
  - generated from output formats 109
  - line leading in 609

- options for OmniHelp 361
- properties, specifying 599
- span class for character formats 602
- using with HTML Help 319
- vs. HTML, header formatting 432
- .css file, cascading style sheet (CSS)
  - options, specifying 593
- css, output format property 118
- curly quotes, converting to straight quotes
  - for print RTF 227
- \$\$\_currbase, macro variable 537, 692
- \$\$\_currfile, macro variable 537, 692
- \$\$\_currfilepath, macro variable 537, 692
- \$\$\_currpath, system-command variable 692, 778
- \$\$\_currtitle, macro variable 537, 692
- custom markers
  - See also* [markers, custom](#)
  - for DITA maps 498
  - for DITA XML 491
  - for DocBook XML 521
  - for HTML extracts 537
  - for WAI
    - image attributes 651
- Cyrillic
  - languages, HTML Help support for 344
  - locale, for index sort order 257
- Czech
  - for HTML Help output, specifying 345
  - for RTF output, specifying 220

## D

- \_d2g\_log.txt, conversion event log file 74
- \_d2g\_log.txt, conversion log file 46
- Darwin Information Typing Architecture, *see* [DITA](#)
- dashed lines, in WMF graphics 746
- Data Type Definition, *see* [DTD](#)
- database input from HTML 443
- dcb, DCL output type 812
- .dcb, output file extension for binary DCL files 73
- dcl, DCL output type 812
- DCL, **DITA2Go** command-line version
  - conversion modules, writing 813
  - files, using existing 73
  - for use with oXygen 36

- options, *listed* 812
- output
  - file extension, specifying 431
  - file structure 813
  - files and paths, specifying 812
  - running 46
  - syntax of 810
- .dcl, output file extension for ASCII DCL files
  - convert existing DCL files 73
- \$\$\_dcount, macro variable 692, 706
- debugging options 75, 821
- default configuration values 61
- definition lists, DITA
  - first-child status 475
  - options for 104
- Delete**, HTML custom marker type 719
- deleting
  - files before conversion 789
- deliverables
  - assembling for distribution 792
  - compiling or archiving 802
  - file names and extensions, *listed* 806
  - producing 788
- dictionary lists, converting
  - to HTML 588
- directory
  - Omni Systems home, creating 29
- directory names
  - restrictions on 26
- distribution
  - assembling files for 792
  - DITA2Go**, downloading 30
- DITA
  - attributes
    - affecting output formats 95
    - assigning to elements 468
    - assigning with conditional flags 169
    - block element, overriding 470
    - block element, specifying 469
    - collection-type 497
    - ID, specifying 468
    - index range, for DITA 1.1 only 456
    - inline element, specifying 470
  - outputclass
    - assigning 471
    - for CSS 459
    - for border and shading properties 88



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- parent, interpolated, assigning 470
- parent, interpolated, overriding 470
- root element, specifying 469
- values of, for output formats 95
- <xref>, overriding 490
- bookmap
  - declarations, overriding 763
- configuration file, custom topic type
  - creating 761
  - listing 763
  - locating 764
- content models
  - See also* content model; DTD
  - built-in, configurations for, *listed* 754
  - built-in, obtaining copies of 754
  - built-in, source of 753
  - configuration files, producing 755
  - configuration sections 756
  - generating from DTDs 754
  - naming 756
  - overriding 762
  - preparing for use 755
- DTD
  - SYSTEM identifier, configuring 457
- elements
  - assigning to formats 461
  - block, ID, specifying 468
  - block, nesting 471
  - default, for character formats 466
  - default, for paragraph formats 462
  - delimiting 460
  - image, configuring 482
  - levels, overriding 479
  - levels, specifying 479
  - list types, parents of 473
  - <menucascade> 467
  - outputclass attribute 467, 471
  - overriding character mapping 466
  - overriding paragraph mapping 463
  - possible parents of, specifying 472
  - root, assigning outputclass attribute 471
  - typographic 467
  - <uicontrol> 467
- images
  - ancestry, specifying 482
  - configuring 482
  - parents of 482
  - wrapping in <fig> elements 483
- maps 763
  - book maps vs. chapter maps 494
- chunking 523
- levels, specifying 495
- naming 494
- navigation aids, providing 498
- navigation title, specifying 495
- nesting 494
- overriding settings with markers 498
- overwriting 493
- predefined markers for, *listed* 498
- titles, specifying 494
- marker types, predefined
  - for maps, *listed* 498
  - for topics and elements, *listed* 492
- parent elements, specifying 472
- producing 455, 493
- project, setting up 455
- PUBLIC declaration 458
- relationship tables
  - adding ALink rows to 497
  - collection-type attribute 497
  - excluding ALink column from 496
  - structure of 496
  - unidirectional linking in 497
- specialized topic types
  - debugging 764
  - element levels 759
  - element parents, specifying 758
  - first-child elements 758
- tables
  - ancestry, specifying 480
  - column widths of, specifying 481
  - empty paragraphs in 481
  - omitting element ancestries 481
  - parents of 480
  - width, specifying 481
- topic ID, specifying 487
- topic types
  - assigned via marker 487
  - assigned via paragraph format 486
  - assignment, precedence of 486
  - default, specifying 486
  - predefined, overriding 762
  - specializing 761
  - specifying 486
- topics
  - alternate titles for 488
  - IDs, specifying 487
  - map levels, specifying 495
  - starting paragraph 485
  - version, specifying 456



**DITA\***, DITA predefined marker types 719  
 for maps, *listed* 498  
 for topics and elements, *listed* 492

## DITA2Go

installing 30  
 running via DCL 809  
 stopping 38  
 uninstalling 38  
 updating 36

## DITA2Map

configuration template for 816  
 map options, specifying 817  
 project, setting up 815  
 running 817  
 understanding 815

dita2map.exe, **DITA2Map** executable 817

## ditamaps

including in output 105  
 individual configuration files for 765  
 output options for, specifying 105

\$\$\_ditastart, macro variable 464

## ditaval files

extracting conditions from 161  
 otherprops values in 162  
 single, specifying 161

divider between topic and related links, inserting 195

## DLL files 37

build numbers of 820  
 downloading 37

.doc files, for Word 2000 237

## DocBook

### attributes

ID, assigning 508  
 inline, assigning 511  
 other than ID, assigning 510  
 overriding 510  
 parent, assigning 511  
 parent, overriding 511

### content model

*See also* content models; DTD  
 built-in  
   configurations for, *listed* 754  
   obtaining copy of 754  
   source of 753  
 configuration file, producing 755  
 configuration sections 756  
 debugging 764

generating from a DTD 754  
 naming 756  
 preparing for use 755

### elements

assigning to formats 504  
 block, ID, specifying 510  
 block, nesting 511  
 default, for character formats 508  
 default, for paragraph formats 505  
 figure, options for 520  
 levels, overriding 519  
 levels, specifying 518  
 list types, parents of 513  
 overriding character mapping 508  
 overriding paragraph mapping 506  
 possible parents of, specifying 512

### images

ancestry, specifying 520  
 figure element, what to include in 520  
 omitting size attributes from 521  
 options for, specifying 520  
 parents of 520  
 titles, where to place 520

language attribute, specifying 434

marker types, predefined, *listed* 521

output, producing 499

parent elements, specifying 512

resources 499

### tables

ancestry, specifying 519  
 parents of 519

**DocBook\***, DocBook predefined marker types 719  
*listed* 521

DocType, specifying for HTML/XML 432

### document

properties, specifying for HTML 438  
 properties, specifying for RTF 134

### double-byte characters

in HTML 434  
 in XML 450

double-byte languages 27, 434

### downloading 37

beta executables 37  
 HTML Help Workshop 32, 315  
 JavaHelp 32  
 Microsoft Help Workshop 33, 281  
 OmniHelp control files 354  
 run-time libraries 37  
 User's Guide 17

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

## DPI

default, specifying for HTML 438

## draft comments

excluding from output 105

specifying formats for 105

<draft-comment>, suppressing in output 94

## drop-down sections

See also [expandable sections for HTML](#)

### blocks for

configuring 271

delimiting with formats 266

delimiting with markers 267

CSS for 271

emulating Web Works Publisher method 275

JavaScript code for 271

### links for

configuring 268

delimiting with formats 266

delimiting with markers 267

## DTD

See also [content model](#)

abstracting content model from 754

HTML, specifying 432

locating for **DITA2Map** 816

parameter entities, equivalent to element sets 758

project, for **DITA2Go**, locating 68

properties, default, changing 180

source for built-in content model 753

**dtd2ini**, content-model extractor 754

duplicate element IDs, checking for 77

dynamic Help systems, *see* [modular Help systems](#)

## E

eBooks, producing 430

from HTML 33

## Eclipse Help

contents and index methods 420

contents properties, configuring 421

context file, naming 426

context-sensitive Help, setting up 425

files, packaging 427

generating 413

index properties, configuring 423

infopops, configuring 425

MANIFEST.MF

configuring 417

including or excluding 416

output options, specifying 414

### plug-in

CSH properties, specifying 420

ID, specifying 417

index properties, specifying 419

naming 416

product version, specifying 417

provider, specifying 417

schema version, specifying 419

TOC properties, specifying 419

### plugin.xml

configuring 418

creating 418

excluding 415

### projects

merging 423

setting up 413

TOCs, primary vs. secondary 423

understanding 413

Eclipse SDK, downloading 33

**EclipseAnchor**, custom marker type 424, 719

**EclipseContext**, custom marker type 426, 719

**EclipseLink**, custom marker type 424, 719

### editor

for log file error display, designating 74

XML, oXygen, integrating **DITA2Go** with 36

eHelp 244

electronic books, producing 430

element names, using for output formats 88

### element paths

assigning **DITA2Go** user variables to 186

defining 92

including in output 72

mapping to output formats 91

inline and block 93

element sets, defining 179

element tags, including in output 72

element types, properties of

*listed* 181

specifying 179, 825

\$\$\_element, macro variable 692

### elements

DITA, configuring 459

DocBook, configuring 504

mapping to formats 87

cross-reference 91

- inline and block 90
  - <steps> headings 88
  - tables 90
  - specialized, defining properties of 179, 825
  - unique IDs for, generating 76
- <\$else>, control structure for macros 704, 705
- <\$elseif>, control structure for macros 704, 705
- embedded topics, DITA
  - IDs for, generated 487
- empty paragraphs
  - in DITA table cells, retaining tags for 481
  - in DocBook table cells, retaining tags for 503
  - in HTML table cells
    - omitting tags for 640
    - providing content for 640
    - retaining tags for 640
  - in HTML text, providing content for 569
  - in RTF output, removing final 228
- end, output format property 118
- <\$endif>, control structure for macros 704, 705
- endnotes, configuring 148
- <\$endrepeat>, control structure for macros 704
- ends, macro string operator 707
- <\$endwhile>, control structure for macros 704
- entity references
  - for HTML 432
  - for XML 450
  - mapped from high ASCII characters 570
- environment variable %OMSYSHOME%, creating 29
- ePub format, producing from XHTML 430
- ePub, producing
  - from HTML output 27, 33
  - from XHTML 430
- error messages
  - DCL NT console driver 821
  - HTML Help
    - alias entries 344
    - page cannot be displayed 318
  - logged as conversion events 74
  - OmniHelp Loading... 819
  - system command, displayed 779
  - text of, localizing for OmniHelp 365
  - Word
    - cannot open file 230
- errors
  - duplicate keys in configuration settings 63
  - logged to conversion log file 74
  - severity level of 74
  - WinHelp compiler 252
- escape character for macros 680
- event log, *see* log file
- events, conversion, logging 74
- excluding content from output 164
- excluding selected elements 94
- expandable sections for HTML 264
  - See also* drop-down sections
  - delimiting with formats 266
  - delimiting with markers 267
  - JavaScript code for
    - deploying 271
    - locating 272
    - modifying 273
  - JavaScript macro for, naming 272
  - understanding 265
- exporting
  - HTML for database input 443
- expressions, macro
  - conditional, in macros 704
  - results of 700
    - displaying in output 702
  - using indirection in 708
  - using list variables in 707
- ExtCode\***, custom marker types 177, 719
- extension point, Eclipse Help 413, 420
- external code snippets, referencing 175
- Extr\***, custom marker types 719
  - listed* 537
- \$\$\_extr\*, predefined macro variables for extracts,
  - listed* 539
- extracts, HTML 528
  - customizing 537
  - delimiting 529
    - with existing formats 529
    - with markers 530
    - with special formats 529
  - enabling and disabling 528
  - meta text for 534
  - naming, with custom markers 782
  - referencing 536
  - replacing with links 539
  - titles of, specifying 531
- <\$extrthumb>, predefined macro 539, 541

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

## F

FAR, for Microsoft Help Viewer 244

favorites option

for HTML Help 320

for JavaHelp2 393

figure anchors, output formats for 102

figure titles

placement of 102

treating as table titles 203

figures

*See also* [graphics](#)

list of (LOF)

file name, specifying 201

generating 201

link to, following related links 191

options for, specifying 102

figures, *see* [graphics](#)

file

*See also* [files](#)

comparison tool, obtaining 35

extension

for DITA XML output 456

for DocBook XML output 501

for graphics, HTML 747

for HTML/XML/DCL output 431

for interfile links 431

for Word interfile links 232

for Word output 219

for XML output 450

extracts, HTML, creating 528, 544

names

containing blanks 812

for DITA topics, via FrameScript 488

for Word interfile references 232

HTML split and extract 530

HTML, custom markers for 782

HTML, generated 198

of chapter-specific configuration files 765

overriding with `copy-to` attributes 524

restrictions on 26

path, *see* [path](#)

paths in configuration settings 64

structure, DCL 813

titles, HTML, specifying 435

**FileName**, custom marker type 719, 782

for DITA topics 487

files

*See also* [file](#)

configuration, *see* [configuration file](#)

converted

default location of for Word 232

copying via system commands 777

deliverable

assembling for distribution 792

default base names of, *listed* 806

generated, naming for HTML output 198

graphics

copied for postprocessing, *listed* 797

extension, specifying for HTML 747

path, removing for HTML 611, 747

replacing, renaming, relocating for HTML 746

Help contents

for HTML Help, generating 334

for JavaHelp, creating 395

for WinHelp, assembling for multiple topics 309

for WinHelp, naming 306

HTML

extracting 528

generated, naming 198

importing as insets 441

renaming, for automated systems 781

split and extract, referencing 536

split and extract, renaming 781

split and extract, specifying titles for 531

splitting 526

splitting, at table heads 526

index

for HTML Help, generating 334

for JavaHelp, generating 395

JavaHelp

helpset, configuring 392

macro, individual 685

macro, library 685

map, HTML Help, specifying 348

naming, restrictions on 26

output, copied for postprocessing, *listed* 795

postprocessing via system commands 777

renaming via system commands 777

splitting

for HTML 526

WinHelp

multiple, referencing from contents 309

project, naming 284

filtering

in maps, advantages of 169

individual elements via mapping 94

- via ditaval files 161
- find, *see* full-text search
- Firefox new window option for OmniHelp 382
- first, macro string operator 707
- \$\$\_firstfile, macro variable 537, 692
- fixed-key configuration sections
  - listed 770
  - overriding settings in 770
  - vs. variable-key 63
- fixed-text links for expandable sections 266
  - configuring 270
- flagging content for further processing 163
- flags
  - See also* conditional, flags
  - conditional, setting 165
  - images for, providing 168
  - properties of, conflicting 168
  - text properties of, specifying 166
- folder, *see* directory
- font
  - RTF, assigning via macro 119
  - size units in CSS 319, 607
  - size, changing in HTML Help 319
  - tags, HTML
    - including in HTML output 578
    - workaround for browser differences 579
- <\$\_fontnum()>, predefined macro for RTF output 119, 684
- fonts
  - mapping, for HTML 576
  - OpenType and TrueType, browser support for 579
- footer rows of tables
  - format for 103
  - in HTML <tfoot> elements 668
  - positioning for HTML 629
- footnotes 582
  - ALink, adding to WinHelp topics 304
  - converting
    - to HTML 581, 603
    - to WinHelp 288
  - inline, configuring, for HTML/XML 582
  - jump, formatting with macros 583
  - links to, eliminating 583
  - omitting
    - from HTML/XML output 582
  - output format of, configuring 102
- separator for 582
- table
  - converting to HTML 603, 642
  - positioning in HTML 642
  - using list tags vs. <div> and <p> tags 583
- format
  - components
    - See also* subformats
    - where to define 141
  - configuration templates, setting up 110
  - mapping options, specifying 87
  - name prefixes, order of 100
  - properties
    - base values of, specifying 120
    - based 116
    - based vs. inline 122
    - before and after, assigning 118
    - block and inline 123, 124
    - css and rtf, applying 118
    - for HTML list formats 125
    - start and end, assigning 118
  - selection paths, including in output 72
  - templates, organization of 730
- format strings in macro expressions 703
- formats
  - See also* character, formats; paragraph, formats
  - character, properties of, overriding 771
  - document section, RTF, configuring 135
  - for table rows, configuring 131
  - list, converting to HTML 584
  - mapping
    - to DITA XML 459
    - to DocBook XML 504
    - to generic XML 452
    - to HTML 565, 590
- output
  - See also* output formats
  - assigning content-adding properties to 118
  - based on other formats 116
  - border properties, block 128
  - configuring 109
  - cross-reference, defining 155
  - cross-reference, specifying 101
  - default, specifying 121, 122
  - document section, RTF, configuring 135
  - for headers and footers, including 120
  - how to define 112
  - inline and block, properties of 123, 124
  - mapping elements to 87
  - modified by attributes 95

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- prefixing with attribute values 95
- properties of, understanding 115
- purpose of 109
- related-link, specifying 192
- RTF section properties, *listed* 135
- table cell properties, *listed* 133
- table cell, configuring 132
- table properties, *listed* 130
- table row properties, *listed* 132
- table row, configuring 131
- table, defining 129
- table, naming and defining 129
- tabs in RTF, specifying 129
- text, naming and defining 121
- where to define 110, 120

## paragraph

- See also* [paragraph, formats](#)
- merging, RTF 226
- properties of, overriding 771
- replacing with code, for WinHelp 287, 711
- replacing with code, for Word 228, 711
- script, designating for HTML 568

- prefixes for, default 100

- properties of, defining 109

fragment identifiers, RFC 5147 177

## framesets

- image maps in 623
- in HTML 443
- in HTML Help 445
- in OmniHelp, customizing 363
- target for HTML jumps 623

framework for Omni Systems applications 29

FTS, *see* [full-text search](#)

## full-text search

- for HTML Help 339
  - excluding topics from 340
- for JavaHelp and Oracle Help 397
- for OmniHelp 367, 371
  - excluding content from 374
  - excluding stop words from 373

# G

## generated lists

- including in HTML output 525
- including links to 190
- naming files for 198

generator, HTML, specifying 435

GhostScript, PostScript interpreter  
for converting EPSI graphics 78

## glossary

- abbreviations for terms, configuring 205
- abbreviations of terms, first-use rules 205
- converting to JavaHelp 401
- file name, specifying 204
- generating 203
- hover text for <glossentry> topics 441
- link to, following related links 191
- overriding default element paths 93

**Graph\***, custom markers for HTML image attributes 720

**GraphAlt**, custom marker for WAI image attribute 651

\$\$\_graphbase, macro variable 616, 692

**GraphDpi**, custom marker for image resolution 720

Graphic Workshop 78

## graphics

- adding space before, HTML 619
- aligning, for HTML 617
- alternate text for
  - HTML 620
  - WAI 650
- aspect ratio, preserving, HTML 621
- attributes
  - See also* [image attributes](#)
  - size, omitting 620
  - specifying 619
  - width and height units 621

## borders around

- eliminating for HTML 619

class, assigning 615

configuration sections subject to override, *listed* 775

## directory

- emptying before copying files to 798
- specifying for assembly 799
- specifying for HTML links 747

## excluding

- from RTF output 751

file extension, specifying, HTML 747

## files

- assembling for distribution 796
- copying to assembly directory 796
- for assembly, listing 797
- path to, on UNIX server 612
- removing path from, HTML 611, 747
- renaming extension, for Bristol Hyperhelp



283  
 replacing, renaming, relocating, for HTML 746  
 thumbnail, naming 540  
 formats, HTML, preferred 746  
 groups  
   assigning properties to 613  
   creating with overrides 775  
   creating, HTML 613, 615  
 in extracts, referencing 542  
 in table cells, repositioning  
   for HTML 619  
 indenting, HTML 617  
 JavaHelp, specifying location of 391, 404  
 names of, including in Word 237  
 properties  
   accessing with <\$\$\_extrgraphid> 542  
   overriding for HTML 752, 774  
   overriding for RTF 752  
   specifying, HTML 619  
 replacement, format for, HTML 747  
 scale, preserving in Word 236  
 scaling  
   for HTML/XML 620  
   for Word 236  
 settings  
   custom, specifying 752  
   synchronizing for HTML output 799  
   synchronizing for RTF output 800  
 size of, preserving, for Word 236  
 spacer, for HTML 617, 641  
 tags around, omitting 616  
 thumbnails, referencing, HTML 540  
 WAI markup for 650  
**GraphLongdesc**, custom marker for WAI image attribute 651  
 \$\$\_graphorighigh, macro variable 616, 692  
 \$\$\_graphorigwide, macro variable 616, 692  
 \$\$\_graphsrc, macro variable 616, 692  
**GraphTitle**, custom marker for WAI image attribute 651  
 Greek  
   **DITA2Go** support for 27  
   for HTML Help output, specifying 345  
   for RTF output, specifying 220  
 groups  
   graphic, *see* **graphics**, groups  
   table, *see* **tables: HTML**, groups

## H

<h1> - <h6>, HTML paragraph tags 566  
 H2reg, for Microsoft Help Viewer 244  
 hard returns  
   in configuration overrides 776  
   to end WinHelp topic titles 298  
 header and footer output formats  
   configuring, for RTF 136  
   including 120  
 headers  
   levels of, for WinHelp 307  
 headers, HTML table attribute for WAI 653  
   purpose of 657  
 headings, localizing 157  
 headings, run-in  
   *See also* **run-in headings**  
   assigning to attribute prefixes 97  
 Helen, third-party JavaHelp compiler 395  
 Help 2, Microsoft, tools for 244  
 Help compiler, WinHelp  
   obtaining 33, 281  
   running automatically 284  
 Help systems, merging 280  
   Eclipse Help 423  
   HTML Help 350  
   JavaHelp, Oracle Help 410  
   OmniHelp 377  
   WinHelp 285  
 Help Viewer, Microsoft  
   index terms for 252  
   tools for converting CHM files 244  
 Help Workshop  
   downloading 33, 281  
   for HTML Help 314  
   for WinHelp 28, 33, 281  
 Help, on-line 243  
   contents entries, configuring 250  
   context-sensitive, setting up 277  
   Eclipse Help  
     evaluating 246  
     generating 413  
   evaluating features of 244  
   HTML Help  
     evaluating 245  
     generating 313  
   HTML-based Help



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- contents levels, setting 251
- index entries, configuring 251
- JavaHelp or Oracle Help
  - evaluating 246
  - generating 385
- merging systems 280
- Microsoft Help 2, tools for 244
- OmniHelp
  - evaluating 245
  - generating 353
- Oracle Help for Java, evaluating 246
- related-topic links, providing 258
- WinHelp
  - contents levels, setting 250
  - evaluating 244
  - generating 281
- HelpMerge**, HTML custom marker type 720
  - for HTML Help 351
  - for OmniHelp 378
- helpset file, JavaHelp, configuring 392
- helpsets, merging 410
- hexadecimal numbers
  - displaying 704
  - in results of expressions 700
- .hha file for HTML Help 343
- .hhc file
  - for HTML Help 336
- .hhp file for HTML Help 318, 320
- HHReg, HTML Help tool 347
- .hht file, CSH IDs for HTML Help 344
- HHW, *see* [HTML Help Workshop](#)
- HIDC\_ prefix for context-sensitive Help IDs 343, 375
- hiding content in Word 228
- hierarchical links in HTML 555
- high ASCII characters
  - encoding for HTML 434
  - encoding for XML 450
  - mapping to HTML 570
  - replacing for W3C validation 445
- highlighting search terms in OmniHelp 374
- home directory, Omni Systems, creating 29
- hotspots
  - HTML Help, span of 264, 322
  - WinHelp
    - defining 299
    - for jumps and pop-ups 299
- hover text, providing, for HTML 441
- .hpj file for WinHelp 282
- htm, DCL output type 812
- .htm, default HTML file extension 431
- HTMConfig**, HTML custom marker type 720
- HTML**
  - See also* [HTML code insertion](#)
  - content for database input 443
  - converting to 429, 611
  - extracts
    - code insertion methods for 534
    - custom markers for 537
    - customizing 537
    - graphics in, referencing 542
    - replacing in parent file 539
    - thumbnails for reference to 540
    - titles, customizing 538
  - file extension, specifying 431
  - files, split and extract, referencing 536
  - generator, specifying 435
  - links, creating 545
  - lists, indenting 589
  - macros for
    - defining and invoking 679
    - including in a library 685
    - selectively enabling 683
    - using expressions in 700
    - using variables in 687
  - navigation macros 555
  - tables, *see* [tables: HTML](#)
  - tags, closing: suppressing 438
  - using XHTML tagging for 431
- HTML code insertion**
  - in splits and extracts 534
  - keyword prefixes for splits and extracts, *listed* 535
  - keywords and locations, *listed* 535
  - keywords for splits and extracts, *listed* 536
  - methods for extracts, *listed* 535
- HTML Help**
  - See also* [HTML-based Help](#)
  - advantages and disadvantages of 245
  - ALink jumps, configuring 261
  - ALinks, target-and-jump 262
  - binary TOC
    - for browse buttons 320, 335
    - mid-topic links 337

- no-link contents entries 337
- browse buttons, enabling 319
- .chm file, specifying 348
- .chm, unblocking 314
- compiling
  - and testing 346
  - for delivery 802
  - with HTML Help Workshop 347
- contents, table of 334
  - customizing 338
  - files, generating 334
  - links to mid-file topics 337
- file name restriction 26
- font resizing 319
- framesets in 445
- full-text search, providing 339
- generating 313
  - contents and index files 334
- href links to other .chm files 324
- hypertext jumps to other .chm files 323
- index entries
  - case sensitivity of, specifying 257
  - levels, combining 253
  - maximum length of 252
  - merging 256
  - sort order, specifying 256
- index files, generating 334
- index, customizing 338
- jumps to secondary windows 333
- KLink jumps, configuring 261
- links
  - specifying syntax of 324
  - to external files, configuring 325
- map files, specifying 348
- merging CHM files 350
- parameters for ActiveX controls 332
- pop-ups
  - creating with HTML Help 322
  - creating with KeyHelp 322
  - creating with WinHelp 323
  - in hypertext **Alert** PI markers 264
- project title, specifying 316
- project, compiling 314
- registering a CHM for network use 347
- related topics, configuring 332
- span of hotspots, determining 264, 322
- starting topic, specifying 317
- synchronizing TOC references 350
- TOC, binary, compiling 321
- uncompiled, configuring links for 324

- viewer
  - for .chm files 314
  - using CSS with 319
- HTML Help Workshop 314
  - downloading 32
- HTML-based Help
  - ALink jumps, configuring 261
  - ALinks, target-and-jump 262
  - contents levels, setting 251
  - index entries
    - case sensitivity, specifying 257
    - sort order, specifying 256
  - index link destination, specifying 255
  - KLink jumps, configuring 261
- HTMLComment**, PI marker type 720
- HyperAlert**, PI marker type 720
- HyperAnchor**, HTML PI marker type 720
- hypergraphic, WinHelp graphic with hotspots 300
- HyperHelp, Bristol, *see* [Bristol HyperHelp](#)
- HyperJump**, HTML PI marker type 720
- HyperLink**, HTML PI marker type 720
- hyperlinks, *see* [hypertext](#), [links](#)
- HyperPopup**, HTML PI marker type 720
- HyperTarget**, HTML PI marker type 720
- hypertext
  - alert** markers, *see* [alert markers](#)
  - links
    - See also* [links](#), [hypertext](#); [links](#), HTML 553
    - for print RTF, converting 231
    - for print RTF, external 231
    - HTML, problem characters in 548
    - WinHelp, using for jumps and pop-ups 301

## I

- icons for drop-down links, configuring 269
- id, HTML table attribute for WAI 653
  - purpose 657
  - via **CellID** marker 666
- identifying
  - Help files and titles, WinHelp 306
  - links to other files, HTML 553
- ideographic space in Japanese HTML Help 346
- IDH\_ prefix for context-sensitive Help IDs
  - for HTML Help 343
  - for OmniHelp 375

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- id/headers method, WAI
  - column and row identifiers, naming 674
  - columns, identifying 674
  - group identifiers, naming 671
  - identifying row and column groups 671
  - markup for table cells 669
  - markup for tables 657, 659
  - rows, identifying 674
  - span identifiers, naming 673
  - using span IDs 675
- IDs
  - DITA element, specifying 468
  - DocBook element, specifying 510
  - duplicate, checking for 77
  - HTML table, *see* TableID
  - symbolic, for HTML Help CSH 341
  - symbolic, for OmniHelp CSH 375
  - unique, building blocks for 76
  - unique, generating for elements 76
- .idx files for Oracle Help 399
- <\$\_if not>, control structure for macros 705
- <\$\_if>, control structure for macros 704
- Illustrator, Adobe, for converting graphics 78
- image attributes
  - See also* graphics, attributes; <img> tag attributes
  - omitting
    - for DITA XML 484
    - for DocBook XML 521
    - for generic XML 453
    - for HTML 620
  - specifying, for HTML 619
- image, background, for HTML 624
- images, *see* graphics
- <img> tag attributes
  - alignment 617
  - class for anchor paragraphs 601
  - specifying 619
    - via markers 650, 722
  - src, specifying 612
    - for JavaHelp 391
- importance attribute, default prefixes and run-in headings for 101
- indenting
  - graphics, HTML 617
  - list items, HTML 126, 589
  - tables, HTML 641
- index
  - See also* index entries
- file, naming 212
- files, generating
  - for HTML Help 334
  - for JavaHelp 395
  - for non-Help HTML 211
  - for OmniHelp 367
  - for Word 207
  - for Word, in Word 240
- heading letters
  - for non-Help HTML 213
- including in single-file HTML output 525
- link destinations for HTML-based Help, specifying 255
- link to, following related links 191
- output format properties, configuring 208
- properties, configuring for Eclipse Help 423
- ranges, including 209
- references, configuring 209
- sort order, specifying
  - for HTML Help, OmniHelp 256
- terms, *see* index entries
- Word
  - configuring 207
  - generating in Word 240
  - including or excluding 221
- index entries
  - See also* index
  - for Eclipse Help
    - configuring 423
  - for Help systems, configuring 251
  - for HTML Help, maximum length of 252
  - for HTML-based Help
    - case sensitivity of 257
    - configuring 251
    - level separators for 252
    - sort order of, specifying 256
  - for JavaHelp
    - configuring 396
  - for Microsoft Help Viewer, preparing 252
  - for non-Help HTML, configuring 212
  - for OmniHelp, configuring 370
  - for Word, configuring 207
- indexes, multiple, configuring 217
- \$\$\_indexfilename, macro variable 692
- indexterms, mapping to variant indexes 217
- index.xml, Eclipse Help index file 423
- indirect references, *see* pointers
- indirection, using in macro expressions 708

infopops for Eclipse Help, configuring 425  
 .ini file, *see* [configuration file](#)  
 inline output format properties, *listed* 123  
 inline text, content-model element type 760  
 inline, output format property 122  
 insets  
     HTML, importing files as 441  
 installing **DITA2Go**  
     for the first time 30  
     updates 36  
 interfile links  
     in HTML, to renamed files 553

## J

Japanese  
     **DITA2Go** support for 27  
     for HTML Help output, specifying 345  
     for HTML Help, compiling 347  
     for RTF output, specifying 220  
     ICU DLLs for HTML output 28  
     ICU DLLs, obtaining 317  
 JAR file, creating 400  
 Java Runtime Environment, for JavaHelp 28  
 Java Virtual Machine, for JavaHelp 385  
 JavaHelp  
     *See also* [HTML-based Help](#)  
     advantages and disadvantages 246  
     ALink jumps, configuring 261  
     compiling, with Helen 395  
     contents and index files  
         creating 395  
         locating 397  
     conversion, setting up directories for 386  
     excluding face attribute of font tags 578  
     full-text search 397  
     generating 385  
     glossary, converting to 401  
     helpset file, configuring 392  
     images, mapping to files 404  
     index entries  
         case sensitivity of, specifying 257  
         configuring 396  
     index link destination, specifying 255  
     JAR file, creating 400  
     JHIndexer command 398  
     map file, specifying location of 391  
     version 2.0, downloading 32  
     windows, defining 403  
 JavaScript  
     for expandable sections 271  
     including in HTML output 27, 430  
     inserting, for HTML attributes 438  
     using macro variables in 536  
**JH2Pop\***, custom markers for JavaHelp 2 pop-up window properties 720  
**JH2Sec\***, custom markers for JavaHelp 2 secondary window properties 720  
 JHIndexer command for JavaHelp 398  
 .jhm file, JavaHelp map file 410  
 JPEG graphics export format  
     for Web use 746  
 JRE, Java Runtime Environment 28  
 jumps  
     ALink  
         configuring for Help systems 261  
         macros for HTML Help 328  
         with keywords for HTML Help 329  
     and pop-ups, WinHelp  
         creating 299  
         using hypertext links for 301  
     destinations of, WinHelp  
         cross-reference, specifying 289  
         external, coding 302  
     KLink, configuring for Help systems 261  
     related-topic, adding for Help systems 261  
     to other Help files, HTML Help 323  
     to secondary windows  
         in Help systems 262  
         in HTML Help 332  
         in OmniHelp 370  
         Oracle Help 408

## K

key names in configuration settings 63  
 Key Tools, obtaining 322  
 keydef  
     limiting scope of 172  
 KeyHelp, DLL for HTML Help pop-ups 322  
 keyref  
     indirection without employing 175  
     to named branch 173  
**KeyrefBranch**, PI for keyref to named map branch

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

173, 720

keyword links, *see* [KLinks](#)

keywords, configuration

DITA content model, *listed* [833](#)

HTML, *listed* [849](#)

RTF, *listed* [837](#)

KLinks

access to merged topics [259](#)

HTML-based Help, configuring [261](#)

jump destinations of, specifying [262](#)

maintenance issues [260](#)

OmniHelp, support for [370](#)

understanding [259](#)

WinHelp, limitations of [303](#)

Korean

**DITA2Go** support for [27](#)

for HTML Help output, specifying [345](#)

for RTF output, specifying [220](#)

## L

label attribute for Eclipse Help index entries [423](#)

label, Eclipse Help TOC, for book level [421](#)

labels, localizing [157](#)

language templates, localizing [157](#)

language, localizing [157](#)

language, output, specifying

for headings, labels, names [158](#)

for HTML Help [344](#)

for <html> tag [433](#)

for print RTF [220](#)

language, overriding [158](#)

languages supported [27](#)

last, macro string operator [707](#)

\$\_lastfile, macro variable [537](#), [692](#)

leading, *see* [line spacing, adjusting](#)

length, macro string operator [707](#)

levels, macro nesting [683](#)

libraries, run-time, downloading [37](#)

library, macro, creating and naming [685](#)

line breaks

in DITA <codeblock> elements, preserving [462](#)

in DITA, inserting via PIs [138](#)

in DocBook <programlisting> elements,

preserving [505](#)

in HTML, suppressing [439](#)

in macros, including or excluding [681](#)

in XML, suppressing [439](#), [451](#)

line spacing, adjusting

for HTML list items [589](#)

for RTF [227](#)

in CSS [609](#)

lines, dashed, in WMF graphics [746](#)

**Link\***, custom markers for HTML link attributes [720](#)

**LinkClass** marker [546](#)

effect of [720](#)

for WAI [652](#)

links

*See also* [cross references](#); [hypertext](#), [links](#)

from maps, types to include [189](#)

hypertext

converting to RTF for Word [231](#)

in output, formatting [192](#)

output format for, specifying [89](#)

peer, resolving [196](#)

related, types to include [190](#)

related-topic

ALinks and KLinks [258](#)

appending to topics [190](#)

for on-line Help [258](#)

to generated lists, appending [190](#)

XML

anchors for, managing [454](#)

configuring [454](#)

links, HTML

configuring [128](#)

creating [545](#)

CSS class, assigning

via format [547](#)

via marker [546](#)

drop-down, configuring [268](#)

buttons [269](#)

icons [269](#)

text [270](#)

type, specifying [268](#)

drop-down, delimiting

with formats [266](#)

with markers [267](#)

for breadcrumb trails [555](#)

forcing to lowercase [549](#)

from cross references [551](#)

hierarchical [555](#)

mid-topic, from TOC [250](#)

- navigation
  - behavior of 559
  - creating 555
  - to footnotes, eliminating 583
  - to other files, identifying 553
- \$\$\_linksrc, macro variable 548, 552, 652, 692
- LinkTitle**, marker for WAI attribute 652
- list formats
  - converting
    - to DITA XML 473, 476
    - to DocBook XML 513
    - to HTML 584
  - CSS properties for, assigning 125
  - dictionary, converting
    - to HTML 588
  - indenting, for HTML 589
  - nested, converting
    - to DITA XML 476
    - to HTML 587
- list variables
  - creating with configuration sections 696
  - for macros 695
  - initializing 696
  - instead of conditional expressions 698
  - processing with macros 696
  - processing with pointers 697
  - using in expressions 707
- lists, definition, options for 104
- lists, generated
  - including in HTML output 525
  - links to, including 190
- lists, parameter, options for 104
- literals, character
  - assigning to macro variables 689
  - for macro variables, *listed* 689
- locale
  - for index sort order 257
  - for RTF output 220
  - identifier for HTML Help 344
  - specifying for HTML Help 344
- localizing headings, labels, names 157
- LocalTOCTitle**, PI marker for HTML local TOCs 720
- log file
  - editor for displaying when errors 74
  - for conversion events 46, 74
- logging

- automation commands 788
- conversion events 74
- logical operators for macro expressions, *listed* 701
- longdesc, HTML image attribute for WAI 650, 651
- loops, nesting, in macros 706
- lower, macro string operator 707

## M

- macro
  - See also* macros, **DITA2Go**
  - configuration file, editing 740
  - expressions, results of
    - displaying in output 702
    - interpreting 700
  - files, individual 685
- macro libraries, organization of 729
- macro parameter, passing 709
- macro templates 187
- macro variables
  - See also* macros, **DITA2Go**
  - assigning paragraph content to 692
  - assigning values to 688
  - assignment values of, displaying 690
  - in HTML navigation macros 560
  - incrementing and decrementing 690
  - list type 695
    - See also* list variables
    - using in expressions 707
  - nesting 690
  - predefined
    - for HTML extract replacement, *listed* 539
    - for HTML splits and extracts, *listed* 537
    - for system commands 778
    - listed* 691
    - uses for 691
  - referenced in WAI attributes 649
  - syntax of 687
  - undefined, debugging 709
  - valid contexts for 710
- \$\$\_macroparam, macro variable 692
- macros, **DITA2Go**
  - See also* macro; macro variables
  - backslash escape character in 680
  - conditional expressions in 704
  - control-structure elements, *listed* 704



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- debugging 709
  - defining 679
  - editing with the Configuration Manager 56
  - expression results 700
  - expressions 700, 709
  - for HTML
    - framesets, using to create 443
    - inserting, for split and extract files 534
    - insertion methods for extracts 534
    - JavaHelp secondary windows and pop-ups 403
    - navigation, inserting predefined 559, 710
    - navigation, redefining 561
    - referenced in WAI attributes 649
    - table, specifying 642
    - using for attribute text 552
    - using for link properties 548
    - using to specify WAI attributes 654
  - line breaks in 681
  - nesting 683
  - nesting limit 683
  - operands 701
  - operators, *listed* 701
  - predefined
    - HTML, *listed* 683
    - RTF, format 119
    - RTF, *listed* 684
  - specifying where to invoke 710
  - ternary operators ? and : 705
  - trailing space in 681
- macros, WinHelp, invoking 302
- manifest file, Eclipse Help
  - MANIFEST.MF, configuring 417
  - plugin.xml, configuring 418
- MANIFEST.MF, Eclipse Help manifest file 416
- map files for context-sensitive Help 278
  - HTML Help, specifying 348
  - JavaHelp, specifying location of 391
- [MAP] section of HTML-based Help file 342
- mapping options for output formats 87
- maps
  - chunking 523
  - ditamaps, configuring 493
  - filtering in, advantages of 169
  - generated, specifying options for 69
  - generating from topics 69
  - including in output 105
  - named branches in, designating 170
  - scoping and filtering in 169
  - specifying output options for 105
- margins
  - of output table cells, default 130
  - of output table cells, specifying 133
  - of RTF output body, specifying 135
  - specifying, for HTML Help pop-ups 322
- marker types
  - See also* [markers](#)
  - configuration, defining 767
  - effects of properties, *listed* 723
  - naming 721
  - predefined
    - for DITA maps, *listed* 498
    - for DITA XML, *listed* 492
    - for DocBook XML, *listed* 521
    - for HTML extracts, *listed* 537
- markers
  - attribute, for HTML or XML
    - for images 619
    - for links 547
    - for tables 634
    - listed* 722
  - configuration, to override settings 767
  - custom
    - for HTML extracts 537
  - hypertext **alert**
    - and **alerttitle**, for WinHelp pop-ups 301
    - for HTML Help pop-ups 264
    - for HTML split points 527
    - for splitting HTML files 527
- memory deallocation 821
- merging
  - Eclipse Help projects 423
  - Help systems 280
  - HTML Help .chm files 348
  - JavaHelp or Oracle Help systems 410
  - OmniHelp projects 377
- Meta\***, PI markers for <meta> tag content 720
- <meta> tag content, supplying 436
  - for split or extract files 534
- Microsoft
  - HTML Help Workshop, *see* [HTML Help Workshop](#); [Help Workshop](#)
  - HTML Help, *see* [HTML Help](#)
  - Vista, no support for WinHelp 244
- Microsoft Help Viewer
  - index terms for 252
  - tools for converting CHM files 244



- mid-topic entry points
  - for Eclipse Help context anchors 426
  - for HTML Help CSH links 343
  - for index links, not recognized by RoboHelp 255
  - for TOC links, in HTML Help 337
  - incompatible with HTML Help binary TOC 320, 337

- mid-topic links
  - from OmniHelp TOC, avoiding 364
  - in Eclipse Help TOC, enabling 422
  - in Help systems, effects of 244

- modular Help systems 280

- modules, DCL conversion, writing 813

- mouseover
  - hover text in HTML output 441
  - shortdesc in title attribute 106
  - for related links 192

- MS HTML Help, *see* [HTML Help](#)

## N

- names
  - See also* [naming](#)
  - of CSS classes, case sensitivity 600
  - of files and paths, restrictions on 26
  - of files, in double quotes 812
  - of formats
    - table, specifying 129
    - text, specifying 121
  - of headings, localizing 157

- namespace, HTML, specifying 433

- naming
  - files
    - and paths 26
    - helpset, JavaHelp 392
    - HTML split and extract 781
    - WinHelp 306
    - WinHelp topic 289
  - formats
    - table 129
    - text 121
  - marker types 721
  - projects
    - Eclipse Help 416
    - OmniHelp 358
    - WinHelp primary window 309

- navigation

- buttons
  - for HTML 560
  - for HTML Help 319
  - for OmniHelp 364
- links for HTML, creating 559
- macros for HTML
  - button definitions, *listed* 564
  - buttons for 560
  - default definitions of 560
  - redefining 561
  - scope of 563
  - text-link definitions, *listed* 563
  - where to invoke 564

- titles
  - for DITA topics, alternate 488
  - from ditamaps, configuring use of 200

- Ndoc, for Microsoft Help Viewer 244

- nested lists
  - converting to DITA XML 476
  - converting to HTML 587

- nested topics, including in TOC 199

- nesting
  - DITA elements 471, 476
  - DITA maps 494
  - DocBook elements 511
  - macro loops and conditionals, forbidden 704, 705
  - macro variables 690
  - macros 683

- network drive
  - for shared configurations 734
  - not a good place for %OMSYSHOME% 29

- network drives
  - why not to use 820

- network file system, using HTML Help across 347
- .new, extension for changed configuration files 51
- <\$\_next>, HTML navigation macro 559
- \$\$\_nextfile, macro variable 537, 692
- \$\$\_nexttitle, macro variable 537, 692
- <\$nopage> index entries
  - for HTML-based Help 255
  - for OmniHelp 370

- no-scroll region for WinHelp topic titles 298

- <note>, default type, specifying 101

- number subformats, properties of, *listed*. 148

- numbered lists, converting
  - to HTML 586

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- to RTF 226
- numbering properties
  - assigning to output formats 150
  - configuring 146
  - of output formats, *listed* 147
  - of output formats, understanding 146
- numeric entity references
  - for XML 450
- numeric IDs for context-sensitive Help 278

## O

- omitting elements from output 94
- Omni Systems
  - environment variable %OMSYSHOME%, creating 29
  - home directory, creating 29
- OmniHelp
  - See also* [HTML-based Help](#)
  - advantages and disadvantages of 245
  - ALink jumps, configuring 261
  - ALink keywords, displaying 370
  - ALinks, target-and-jump 262
  - buttons, excluding or displaying 364, 369
  - contents
    - expanding and collapsing 368
    - including 367
  - context-sensitive Help, setting up 375
  - cookies, persistence of 381
  - CSS usage, specifying 361
  - data and control files, *listed*
    - supplied in ohview.zip 356
    - generated by **DITA2Go** 357
  - files, obtaining 354
  - frameset and frame dimensions, specifying 363
  - full-text search
    - configuring 371
    - including 367
    - terms, highlighting 374
  - index entries
    - case sensitivity of, specifying 257
    - expanding and collapsing 368
    - levels, combining 253
    - See and See also* entries 370
    - sort order, specifying 256
  - index link destination, specifying 255
  - index, including 367
  - interface, localizing 246, 365
  - KLink jumps, configuring 261
  - launching 381
  - memory requirements 359
  - navigation aids, modifying 364
  - navigation panel, modifying 365
  - pop-up windows, specifying 371
  - prev/next buttons, including 364
  - projects
    - merging 377
    - naming 358
    - setting up 357
    - titles of 359
  - related topics
    - including 367
    - links, providing 370
  - search terms, highlighting 374
  - secondary windows, jumping to 370
  - settings, making persistent 365
  - starting topic, specifying 359
  - template, modifying 366
- %OMSYSHOME% environment variable 29
- on-line Help, *see* [Help, on-line](#)
- OpenOffice, producing RTF for 241
- operating settings, specifying 67
- operators for macro expressions, *listed* 701
- Oracle Help for Java
  - See also* [HTML-based Help](#)
  - advantages and disadvantages of 246
  - ALink jumps, configuring 261
  - ALinks, target-and-jump 262
  - content and index, creating 397
  - Developer's Kit 28
  - downloading 32
  - full-text search 397
  - index entries
    - case sensitivity of 257
    - configuring 396
  - index link destination, specifying 255
  - JAR file, creating 400
  - obtaining information about 385
  - windows, defining 403
- order of configuration-file sections and settings 63
- otherprops values, complex, processing 162
- output 131, 135
  - directory, specifying 39
  - file paths and names, specifying 812
  - numbering properties, configuring 146
  - print, including or excluding content for 71
  - text strings, configuring 141

- type, specifying 39
  - for print RTF 219
  - for WinHelp 284
- output formats
  - See also* [formats, output](#)
  - block properties, *listed* 124
  - border properties, block 128
  - configuring 109
  - cross-reference, default names of, *listed* 157
  - cross-reference, specifying 101
  - default names for 88
  - default, specifying 121, 122
  - footnote properties 102
  - for cross-reference elements 91
  - for links 89
  - for unused features 120
  - from @outputclass 88
  - from element names 88
  - how to define 112
  - index properties 208
  - inline and block, properties of 123, 124
  - inline properties, *listed* 123
  - mapping elements to 87
  - mapping options 87
  - modified by attributes 95
  - numbering properties of, *listed* 147
  - numbering properties of, understanding 146
  - properties of, based on other formats 116
  - properties of, understanding 115
  - purpose of 109
  - RTF section properties, *listed* 135
  - table cell properties, *listed* 133
  - table cell, configuring 132
  - table row properties, *listed* 132
  - table, naming and defining 129
  - table, properties of, *listed* 130
  - text, naming and defining 121
  - text, understanding 119
  - where to define 110, 120
- outputclass attributes
  - for border and shading properties 88
  - for output format names 88
  - mapping values to output formats 89
    - for block and inline elements 90
    - for cross references 91
    - for tables 90, 103
  - of indexterm elements, mapping to variant indexlists 217
- overline, replacing with a tag in HTML/XML 580

- overrides
  - See also* [overriding](#)
  - configuration
    - See also* [configuration settings, overriding](#)
    - for HTML table and graphics groups 775
    - persistent vs. temporary 767
  - format
    - allowing or eliminating for HTML 573
    - suppressed for DITA XML 467
- overriding
  - configuration settings
    - fixed-key 770
    - in macros 767
    - variable-key 771
    - with command-line options 810
    - with configuration markers 767
    - with text, for HTML 776
  - cross-reference properties 773
  - format properties 771
  - HTML graphics properties 774
  - HTML table
    - [Attributes] values 644
    - column and row groups 630
    - default heading/footer counts 631
    - default WAI cell settings 675
    - display attributes 633
    - properties 773
    - WAI markup method 659
  - paragraph properties for HTML 570
  - path to graphics for HTML 748, 774
  - split points in HTML 527
- overview topic in WinHelp 309
- oXygen, integrating **DITA2Go** with 36

## P

- </p> tags, suppressing in HTML 438
- padding
  - HTML, around body text 120
  - of output table cells 133
  - of output table cells, default 130
- page
  - breaks
    - between title and TOC 525
    - forcing, for RTF 137
    - handling for WinHelp 285
    - inserting in DITA via PIs 137, 138
  - header and footer output formats, configuring for RTF 136

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- layouts for RTF, defining 134
- numbers
  - in cross references, for print RTF 231
  - including in cross-reference formats 155
- titles, for HTML files 531
  - assigned with markers 533
  - based on file names 533
  - based on paragraph formats 532
  - computed 534
  - default 533
  - precedence 531
  - prefixes and suffixes 532
- PAGEREF field, RTF, assigning via macro 119
- <\$pageref>, predefined macro for RTF output 119, 684
- Paint Shop Pro, for converting graphics 78
- paragraph
  - See also* paragraphs
  - attributes, suppressing, for HTML 568
  - autonumbers, eliminating, for HTML 454, 567
  - formats
    - See also* formats, paragraph
    - eliminating tags for HTML 568
    - for splitting HTML files 526
    - mapping to DITA elements 461
    - mapping to DocBook elements 504
    - mapping to RTF styles 225
    - merging, RTF 226
    - output properties, *listed* 123
    - output, default, specifying 121, 122
    - properties of, overriding 771
    - replacing with code, for WinHelp 287, 711
    - replacing with code, for Word 228, 711
    - replacing with comments for HTML 568
    - script, designating, HTML 568
  - properties
    - changing for individual paragraphs 771
    - overriding, HTML 570
    - stripping, HTML 568
- paragraphs
  - See also* paragraph
  - empty
    - providing content for in HTML 569
  - replacing with RTF code
    - for WinHelp 287, 711
    - for Word 228, 711
  - unwanted, eliminating for HTML 569
- parameter entities, equivalent to element sets 758
- parameter for DITA2Go macro 709
- parameter lists
  - for DITA output 475
  - options for DITA input 104
- \$\$\_paratag, macro variable 692
- pass-through code, HTML 568
- path
  - See also* file, paths in configuration settings
  - current, macro variable for 537
  - default, for Word documents 232
  - element, mapping to output formats 91
  - names
    - restrictions on 26
    - spaces in, avoiding 26, 820
  - omitting from links, for OmniHelp 359
  - overriding, for HTML graphics 748, 774
  - relative vs. absolute
    - in configuration settings 64
    - in graphics references 612
  - retaining in interfile links for HTML 553
  - specifying, for HTML graphics 612
  - to assembly directory 792
  - to configuration template 731
  - to CSS directory, for copying CSS files 800
  - to graphics files
    - on UNIX server 612
    - removing, for HTML 611, 747
  - to project directory, macro variable for 692
  - to project DTD, specifying 68
  - to shipping directory 806
  - to XML catalogs, specifying 68
- PDF output, producing via Word 222
- peer links, resolving 196
- pernicious mixed content in DITA source 181
- persistent configuration overrides 767
- persistent settings in OmniHelp 365
- pictures, *see* graphics
- pkzip.exe
  - for archiving deliverables 803
  - for packaging Eclipse Help topic files 427
- placement of, for HTML/XML 582
- platform differences, accommodating, WinHelp 283
- plug-in manifest file plugin.xml, Eclipse Help, configuring 418
- plugin.xml, Eclipse Help manifest file 418
- pointers
  - to process lists 697
- policy, chunking, specifying 524

## pop-ups

- See also* [windows, pop-up](#)
- browser, suppressing, effect on OmniHelp [383](#)
- HTML Help [322](#)
  - creating with HTML Help [322](#)
  - creating with KeyHelp [322](#)
  - creating with WinHelp [323](#)
- HTML, require JavaScript [550](#)
- JavaHelp, using macros for [403](#)
- OmniHelp, specifying [371](#)
- WinHelp
  - alert, creating [301](#)
  - creating [299](#)
  - from table cells [292](#)
  - hotspots for [299](#)
  - using hyperlinks for [301](#)

## postprocessing

- activating and logging [788](#)
- automated [787](#)
- files copied, *listed* [795](#)
- graphics files copied, *listed* [797](#)
- separately from converting [807](#)
- understanding [787](#)

`<pre>`, HTML paragraph tag [566](#)

## precedence

- of configuration settings [732](#), [742](#), [765](#)
  - listed* [766](#)
- of DITA topic type assignments [486](#)
- of extract code insertion methods [534](#)
- of extract property assignments [538](#)
- of HTML page title assignments [531](#)
- of macro definitions [683](#), [687](#)
- of macro variable definitions [688](#)
- of table property assignments [625](#)

## predefined

- macro control-structure elements, *listed* [704](#)
- macro variables
  - all, *listed* [691](#)
  - for HTML splits and extracts, *listed* [537](#)
  - in system commands [778](#)
  - using [691](#)
- macros
  - for HTML, *listed* [683](#)
  - for RTF formats [119](#)
  - for RTF, *listed* [684](#)

## prefixes, format-name

- assigning run-in headings to [97](#)
- default, from attributes [100](#)
- order of [100](#)

## preformatted text

- assigning HTML `<pre>` tags [566](#)
- content-model element type [760](#)
- HTML/XHTML, configuring [581](#)
- in table cells
  - for DITA [481](#)
  - for DocBook [503](#)
  - for HTML/XML [640](#)
- preserving whitespace in for DITA [471](#)
- `<$_prev>`, HTML navigation macro [559](#)
- `$$_prevfile`, macro variable [537](#), [692](#)
- `$$_prevtitle`, macro variable [537](#), [692](#)
- primary window, naming in WinHelp [309](#)
- print, output to be included or excluded for [71](#)
- Print**, PI marker [720](#)
- Print**, PI marker for conditional output [72](#)
- `$$_prjpath`, system-command variable [692](#), [778](#)

## project file

- HTML Help [318](#), [320](#)
- JavaHelp helpset [392](#)
- WinHelp [282](#)
  - naming [284](#)

## Project Manager, **DITA2Go**

- shortcut to [31](#)
- to create a project [39](#)

## project, **DITA2Go**

- creating [39](#)
- DTD, specifying [68](#)
- naming [39](#)

## properties

- of element types
  - assigning to class attributes [183](#)
  - default [825](#)
  - defining [179](#)
- of formats
  - based vs. inline [122](#)
  - block, specifying [124](#)
  - border [128](#)
  - character, specifying [121](#)
  - content-adding [118](#)
  - css and rtf, assigning [118](#)
  - implied, for HTML links [128](#)
  - inline, specifying [123](#)
  - list, **DITA2Go** list styles [127](#)
  - list, HTML list styles [125](#)
  - paragraph, specifying [122](#)
  - table [129](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

public and system IDs, overriding 763

PUBLIC declaration for HTML/XML 432

punctuation

in ALink keywords, avoiding

for HTML Help 326

for OmniHelp 370

for Oracle Help 409

in CSH **newlink** markers

for JavaHelp, Oracle Help 410

in file and directory names, avoiding 26

in HTML file names 783

in index entries

ignoring for sort order 256

in link keywords, disallowed 258

pushatstart, pushatend

conref push actions 176

for conref between topics 176

## Q

quotes

around configuration-assignment values 768

around macro names in overrides 768

style, specifying for print RTF 227

style, specifying for <q> output 143, 159

## R

redirect pages for OmniHelp CSH 376

.ref files, interfile links

when not to delete 791

reference files for HTML

deleting between conversions 790

related links

appending to topics 190

changing for peers 196

divider, configuring 195

generating ALinks from 192

including descriptions with 191

labels for, specifying 193

listing by topic type 191

output appearance of 193

output formats for, specifying 194

related topic

keywords

adding with format properties 260

adding with markers 260

links

ALinks and KLinks 258

for Help systems 258

for HTML Help 325

for OmniHelp 367, 370

for Oracle Help 409

for WinHelp 303

in DITA maps 496

relational operators for macro expressions, *listed* 701

relationship tables, DITA 496

relative vs. absolute paths

in configuration settings 64

in graphics references 612

renaming files via system commands 777

repeat loops in macros 706

<\$\_repeat>, control structure for macros 704, 705

replace with, macro string operator 707

requirements, system 27

resource.h, Help map file 278

returns, hard, *see* **hard returns**; **line breaks**

reusing topics 170

revision tracking in Word 239

RFC 5147 fragment identifiers 177

RGB colors

Web-safe 440

*listed* 441

RoboHelp, for generating WebHelp 244, 245, 255

root element

for content model 757

row output formats for tables, configuring 131

row spans, identifying 673

**Row\***, custom markers for HTML or XML table row attributes 720

**RowClass**, custom marker for CSS 635

RowGroup cells 677

and ColGroup cells, using 676

with id/headers method 670

with scope attributes 668

*defined* 661

rowspan, HTML table attribute 653

RTF

color number, assigning via macro 119

converting to 219

formats, mapping 225



- output macros, predefined 119
- PAGEREF field, assigning via macro 119
- raw code, replacing content
  - in WinHelp 287, 711
  - in Word 228, 711
- style name, assigning via macro 119
- STYLEREF field, assigning via macro 119
- tab positions, specifying 129
- RTF code, including for Word 238
- rtf, DCL output type 812
- rtf, output format property 118
- RTFConfig**, RTF custom marker type 720
- run-in headings
  - assigning to format-name prefixes 97
  - configuring 153
- Russian
  - DITA2Go** support for 27
  - for HTML Help output, specifying 345
  - for RTF output, specifying 220

## S

- SAppLocale, HTML Help compiler for other locales 347
- scaling
  - graphics for HTML/XML 620
  - thumbnail graphics for HTML 541
- scope method, WAI
  - identifying column and row groups 668
  - identifying columns and rows 668
  - markup for tables 657, 658
- scope, HTML table attribute for WAI 653
  - adding via format 662, 663
  - adding via marker 666
  - purpose of 657
- script paragraph formats, designating, HTML 568
- <script>, HTML paragraph tag 566
- scrolling WinHelp topic titles 298
- Search**, PI marker 340, 374, 720
- search, *see* full-text search
- secondary windows
  - See also* windows, secondary
  - HTML Help 332
    - accessing from contents or index 333
    - accessing from topics 333
    - defining 333

- JavaHelp
  - size and position settings for 403
  - using macros for 403
- jumping to, in Help systems 262
- OmniHelp, specifying jumps to 370
- WinHelp
  - forcing contents to main window 309
  - specifying 302
- see* and *see-also* index properties, configuring 208
- separator character
  - between index references 210
  - between menu items 159
  - between topics, adding 525
  - in file paths 64
  - for importing HTML files 441
  - in system commands 778
  - in index ranges 210
- SEQ fields, Word, for autonumbers 226
- service mark format, configuring 157
- sets, element, defining 179
- setting up a conversion
  - to DITA XML 455
  - to DocBook XML 500
  - to Eclipse Help 413
  - to generic XML 449
  - to HTML 430
  - to HTML Help 315
  - to JavaHelp or Oracle Help 386
  - to OmniHelp 357
  - to print RTF 239
  - to WinHelp 281
- ShadeFormat**, PI marker for specifying shading subformats 720
- ShadeType**, PI marker for specifying shading subformats 146, 720
- shading
  - for output formats, from <outputclass> attributes 88
  - formats, defining 145
- shed.exe, for WinHelp graphic hotspots 300
- .shg files, WinHelp hypergraphics 300
- \_ship, default shipping subdirectory 247, 788
- shipping directory, specifying 806
- shortcut to ugdita2go.chm, creating 31
- shortdesc
  - deciding where to display 107
  - including in title attribute



# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

- as a mouseover 106
  - for related links 191
  - including in TOCs 200
  - including with links 192
- SimpleTableRelCol**, PI marker for table column widths 720
- SimpleTableWidth**, PI marker for table width 720
- smart quotes, converting to straight quotes
  - for print RTF 227
- solidus, mapped to a forward slash for HTML 576
- sort order, index, specifying 256
  - See also* [index](#), [sort order](#), [specifying](#)
- source directory, specifying 39
- source map file, selecting 39
- space, adding
  - See also* [line spacing](#), [adjusting](#)
  - at the end of a macro 681
  - before graphics in HTML 619
  - before tables in HTML 643
  - between topics in a single HTML file 525
- spacer between fixed links, configuring 195
- spacer graphic for HTML
  - for indenting images 617
  - for indenting tables 641
- \$\$spacerwidth, macro variable for HTML 685
- spaces
  - around images in HTML table cells,
    - eliminating 619
  - fixed, in Japanese HTML Help 346
  - in configuration settings 63, 73
  - in CSS class names, removing or replacing 600
  - in file or path names
    - for commands, double quotes 812
    - for HTML links, avoiding 554
    - not recommended 26, 820
  - in HTML links, removing or replacing 548
  - removing from a string value 709
  - trailing, in macros 681
- span class, CSS attribute for character formats 602
- span, HTML table attribute for WAI 719
- special characters, mapping for HTML 574
- specializations, providing XML catalogs for 68
- specialized elements, defining properties of 179, 825
- splash screen, break before TOC 525
- split files
  - See also* [split points](#)
  - HTML 526
    - designating split points for 526
    - naming
      - via paragraph formats 782
      - via PI markers 782
    - suppressing split points for 527
    - titles of, specifying 531
- split points
  - See also* [split files](#)
  - for DITA XML files 484
  - for HTML files
    - overriding 527
    - preventing dangling headings with 527
    - suppressing 527
    - using Help-contents level numbers for 527
- Split**, PI marker for splitting files 526, 720
- StarOffice, producing RTF for 241
- start, output format property 118
- starting configuration file, copied to project directory 39
- starting topic, specifying
  - for Eclipse Help 421
  - for HTML Help 317
  - for JavaHelp 393
  - for OmniHelp 359
  - for Oracle Help 393
  - for WinHelp 306
- starts, macro string operator 707
- stop words in OmniHelp search 373
- stopping a **DITA2Go** conversion 38
- straddled table columns and rows
  - in WinHelp 292
  - in Word 233, 234
- strikethrough, as a format override for HTML 573
- string operators for macro expressions, *listed* 702
- stripping paragraph properties for HTML 568
- structure, XML, providing 451
- style tags, HTML/XML, suppressing in output 567, 570
- style, RTF, assigning via macro 119
- <\$\_style(>), predefined macro for RTF output 119, 684
- <\$\_stylenum(>), predefined macro for RTF output 119, 684
- STYLEREF field, RTF, assigning via macro 119

<\$\_styleref()>, predefined macro for RTF  
output 119, 684

stylesheet, Word, generated from output formats  
109

subformats  
aliases for, assigning 142  
properties of, based 142  
where sought 142  
where to define 141

suffix, file name  
for generated index 212  
for glossary 204  
for list of figures 201  
for list of tables 202  
for table of contents 199

suffix, file, *see* file, extension

summary, HTML table attribute for WAI 653, 654

support for **DITA2Go**, requesting 819

suppressing elements 94

symbolic IDs for context-sensitive Help 278

syntax  
command-line  
for DCL 810  
configuration-variable assignment 768  
macro variable, for HTML 687

system  
commands, *see also* commands, system  
commands, to automate conversions 777  
requirements for **DITA2Go** 27

## T

table cells  
HTML, *see* tables: HTML, cells  
format properties for  
configuring 132  
*listed* 133

table formats, naming and defining 129

table of contents, *see* contents; TOC

table structure model, CALS vs. HTML 627

table titles, from figure titles 203

**Table\***, PI markers for HTML table attributes 720

TableID  
assigning properties to, for HTML 626

tables, format properties for  
configuring 130

*listed* 130

tables, list of (LOT)  
file name, specifying 202  
generating 202  
link to, following related links 191

tables, options and formats for 103

tables: converting  
to HTML 625  
to WinHelp 290  
to Word 232

tables: HTML  
access method, specifying for WAI 658  
adaptive sizing of 639  
attributes  
automatically generated, eliminating 453  
overriding 630, 633, 639  
attributes, specifying  
precedence of methods 625  
via [Attributes] 632  
via macros 642  
via markers 634  
background color, automatic 635  
browser-dependent tags for 628  
caption tags 641  
cells  
attributes of, specifying 645  
format properties of 132  
identifying 664, 674  
column groups  
enumerating 628  
identifying 628  
overriding 630  
columns  
applying CSS class attribute to 629  
WAI information about 657  
configuration sections subject to override, *listed*  
774  
converting to paragraphs 647  
display attributes  
overriding 633  
properties for, specifying 632  
specifying 632  
footer rows, counting 631  
footnotes  
converting 642  
positioning 642  
graphics in, adjusting spacing 619  
groups  
assigning properties to 626

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

- creating 626
  - creating with overrides 775
  - specifying settings for 625
  - using wildcards to specify 627
- header cells, designating 628
- header rows
  - counting 631
  - designating 670
- indenting 641
- macros, specifying 642
- properties of, overriding 773
- properties, assigning 625
- row groups
  - attributes of, specifying 644
  - identifying 628
  - overriding 630
  - specifying 629
- rows
  - attributes of, specifying 634, 644
  - information for WAI 657
- space before, adding 643
- splitting files based on 526
- structure, specifying 627
- titles, positioning 641
- WAI markup 652
  - applying 652
  - method for, choosing 653
  - method for, default, specifying 658
  - overriding 659
  - strategy for 657
- tables: WinHelp
  - adaptive sizing of 291
  - appearance, adjusting 291
  - converting rows to topics 292
  - titles, positioning 291
- tables: Word
  - cell properties, adjusting 233
  - titles, repositioning 233
- TableSummary**, PI marker for WAI 655
- TableTitle**, PI marker for WAI 655
- tabs
  - in autonumbers, eliminating for HTML 568
  - in configuration settings 63
  - in output formats, specifying for RTF 129
- tags, HTML/XML, eliminating from output 567, 570
- target frame for HTML jumps 623
- \$\$\_tblcols, macro variable 645, 692
- \$\$\_tblrows, macro variable 645, 692
- <tbody> elements
  - and RowGroup cells 677
  - overriding [Attributes] for 644
  - required for scope row groups 668
  - tags for HTML tables 629
- technical support for **DITA2Go**, requesting 819
- template macros, **DITA2Go** user variables in 187
- templates
  - configuration 727
    - chaining 743
    - creating 741
    - format, organization of 730
    - format, setting up 110
    - general, organization of 728
    - general, what to include in 742
    - naming convention for 728
    - organization of 727
    - precedence of 743, 765
    - referencing 731
  - language, for headings, labels, names 157
  - OmniHelp, modifying 366
  - Word, specifying 223
- temporary configuration overrides 767
- ternary macro operators '?' and ':' 705
- test file title, eliminating 435, 531, 533
- text
  - color, specifying
    - for HTML 580
    - for HTML or RTF 123
  - for drop-down links, configuring 270
  - formats, naming and defining 121
  - output, literal, configuring 141
  - pop-up attributes, HTML Help 322
  - preformatted
    - configuring, HTML 581
    - designating, HTML 566
  - replacing, with code or macros 711
- <tfoot> elements
  - overriding [Attributes] for 644
  - tags for HTML tables 629
- <th> elements, tags for HTML tables 628
- <thead> elements
  - overriding [Attributes] for 644
  - tags for HTML tables 629
- thumbnails to reference graphics
  - in HTML extracts 540
  - in place of images in HTML 616

- title page
  - break before TOC [525](#)
  - for RTF output, configuring [138](#)
- Title**, custom marker for split and extract files [720](#)
- title, DITA attribute
  - including shortdesc in [106](#), [191](#)
- title, HTML attribute
  - for images
    - assigning via format [650](#)
  - for links
    - assigning via format [652](#)
    - assigning via marker [651](#)
  - for tables
    - assigning via TableID [654](#)
    - WAI guidelines for [653](#)
- Title, HTML marker type property [533](#)
- titles
  - DITA, alternate, specifying [488](#)
  - HTML Help project, specifying [316](#)
  - HTML, specifying [435](#)
    - for split and extract files [531](#)
    - to eliminate *Test File* [435](#), [531](#), [533](#)
  - JavaHelp helpset, specifying [392](#)
  - WinHelp
    - file, identifying [306](#)
    - table, repositioning [291](#)
    - topic, configuring [297](#)
- TOC
  - entries from <navtitle> elements [200](#)
  - generated from map [198](#)
  - in single-file HTML output [525](#)
  - including shortdesc in [200](#)
  - page break before [525](#)
- TOC, generating [198](#)
- toc.xml, Eclipse Help TOC file [421](#)
- <\$\_top>, HTML navigation macro [559](#)
- topic
  - See also* [topics](#)
  - DITA, starting point of [484](#)
  - files, WinHelp
    - assembling contents for [309](#)
    - naming [289](#)
  - ID, DITA, specifying [487](#)
  - levels in WinHelp, specifying [307](#)
  - starting, specifying
    - for Eclipse Help [421](#)
    - for HTML Help [317](#)
    - for JavaHelp [393](#)
    - for OmniHelp [359](#)
    - for Oracle Help [393](#)
    - for WinHelp [306](#)
  - titles in WinHelp, configuring [297](#)
  - type, DITA
    - default [486](#)
    - specifying [486](#)
- TopicAlias**, custom marker for context-sensitive help [720](#)
- topicref
  - pull from external map [175](#)
- topics
  - See also* [topic](#)
  - DITA, *see* [DITA](#), [topics](#)
  - pop-up, *see* [pop-ups](#); [windows](#), [pop-up](#)
  - reusing via map branches [170](#)
  - WinHelp
    - adding ALink footnotes to [304](#)
    - converting table rows to [292](#)
    - creating [294](#)
- TopicStartCode**, PI marker for code at start of topic [683](#), [720](#)
- <\$\_TopicStartCode>, predefined macro for HTML [683](#)
- trademark formats, configuring [157](#)
- <\$\_trail>, predefined macro for HTML [555](#), [683](#)
- trailing space, in macros [681](#)
- trails of links, creating for HTML [555](#)
- trim first, macro string operator [707](#)
- trim last, macro string operator [707](#)
- truncating cross-reference marker text, WinHelp [290](#)
- Turkish
  - for HTML Help output, specifying [345](#)
  - for RTF output, specifying [220](#)
- type attribute
  - default prefixes and run-in headings for [101](#)
  - of <note> element, default value of [101](#)
- type of, for non-Help HTML [211](#)
- typographic elements
  - assigning to a format for DITA output [467](#)
  - including for DITA XML [457](#)
  - managing in HTML/XML [579](#)
  - replacing with other tags [580](#)
  - suppressing in HTML/XML
    - all [580](#)
  - use sparingly for DITA XML [466](#), [467](#)

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

typographic tags  
     for text output 143  
     nesting 143

## U

unblocking CHM files 314  
 underlined text  
     for hotspots in WinHelp 296, 300  
     for links in HTML 545  
     solid vs. dotted, for WinHelp hotspots 299  
 underscores  
     allowed in WinHelp reference strings 297  
     avoiding in path and file names 26  
         for HTML Help 553  
     disallowed in CSS class names 600  
     disallowed to start user variable names 688  
     removing from path and file names 820  
     replacing spaces in graphics file names 748  
 Unicode  
     character ranges, assign CSS classes to 603  
     characters, mapping in HTML 574  
     conversion for HTML 434  
     decimal value for character mapping 575  
     space after, in RTF 221  
 unique IDs  
     building blocks for 76  
     generating for elements 76  
 UNIX server, relative path to graphics 612  
 <\$\_until>, control structure for macros 704, 705  
 upper, macro string operator 707  
 user variables, *see* variables, DITA2Go user  
 UTF-8 character encoding 450

## V

validating HTML documents 434, 445  
 valign and align, automatically generated, excluding from HTML table cells 453, 635  
 variable-key configuration sections  
     for cross-reference formats, *listed* 773  
     for HTML graphics properties, *listed* 775  
     for HTML table properties, *listed* 774  
     for text formats, *listed* 772  
     vs. fixed-key 63  
 variable-key settings, overriding 771

variables, *see*:  
     variables, DITA2Go configuration  
     variables, DITA2Go macro  
     variables, environment  
     variables, DITA2Go user  
 variables, DITA2Go configuration  
     assigning macros and variables to 768  
     assigning values to 768  
     capturing settings with 699  
 variables, DITA2Go macro  
     *See also* macro variables 692  
     assigning paragraph content to 692  
     assigning values to 688  
     incrementing and decrementing 690  
     list, using in expressions 707  
     list, working with 695  
     nesting 690  
     predefined  
         all, *listed* 691  
         for extracts, *listed* 539  
         for splits and extracts, *listed* 537  
     starting values for 689  
 variables, DITA2Go user  
     deploying in template macros 187  
     element paths, assigning to 186  
     including in macros 186  
     predefined, in system commands 778  
     single- vs. multiple-instance, accessing 185  
 variables, environment  
     %OMSYSHOME%, creating 29  
 variables, user *see* variables, DITA2Go user  
 version of DITA2Go  
     command-line 809  
     how to find 820  
 view output command 34  
     for Help systems 248  
     for Word 223  
 Vista, Microsoft, support for WinHelp 244

## W

W3C  
     HTML 4 specification 432  
     placement of <tfoot> elements 629  
 WAI  
     abbr attribute  
         assigning to a paragraph format 662  
         assigning with a special marker 666



- ul style="list-style-type: none;">
- assigning with a special paragraph 666
- attributes
  - assigning to paragraph formats 650, 661
  - assigning values to 663
  - comparing ways to specify 649
  - for links 651
  - image 650
  - image, assigning to a paragraph format 651
  - image, PI markers for 651
  - specifying with paragraph formats 649, 650
  - supplying as paragraph content 650
  - using special paragraphs for 665
- axis attribute
  - assigning to a paragraph format 662
  - assigning with a special marker 666
  - assigning with a special paragraph 666
- cells
  - header, group properties of 660
  - identifying 664
  - identifying by row and column 674
  - overriding default settings 675
  - tags for, assigning with paragraph formats 663
- ColGroup cell, *defined* 660
- column groups, defining 660
- guidelines
  - for images 650
  - for links 652
  - for tables 653
- id attribute, assigning with a special marker 666
- id/headers method for table cells 669
- link attributes, assigning to a paragraph format 652
- markup
  - for images 650
  - for links 651
  - for tables 652
- row groups, defining 661
- RowGroup cell, *defined* 661
- scope attribute
  - assigning to a paragraph format 662
  - assigning with a special marker 666
- span attributes 672
- summary attribute 654
- table markup, *see* tables: HTML, WAI markup
- title attribute 654
- warnings, logging as conversion events 74
- watermark, as background image for HTML 624
- \$\$\_wcount, macro variable 692, 706
- Web Accessibility Initiative, *see* WAI
- Web browsers, *see* browsers
- Web Works Help 244
- WebHelp
  - evaluating 245
  - from HTML Help and RoboHelp 244, 255
- Web-safe colors, *see* colors, Web-safe
- while loops in macros for HTML 705
- <\$\_while>, control structure for macros 704, 705
- whitespace, preserving
  - in DITA block elements 471
  - in HTML output 581
- wildcards, using
  - in attribute values 163
  - in configuration settings 65, 73
  - in ditaval statements 161
  - in HTML special-character mappings 574
  - to identify tables for HTML 626
  - to specify table sets for HTML 627
- window
  - browser, opening another 444
  - JavaHelp main, naming 403
  - WinHelp main, naming 309
- Window**, PI marker for HTML Help secondary windows 720
- Windows Registry
  - browser command for OmniHelp CSH calls 376
  - CHM files, registering 347
- windows, pop-up
  - See also* pop-ups
  - HTML Help 264, 322
  - HTML, require JavaScript 550
  - JavaHelp 264, 403, 406
  - OmniHelp 264, 371
  - Oracle Help 264, 408
  - WinHelp 299
- windows, secondary
  - See also* secondary windows
  - HTML Help 332
    - defining 333
    - jumping to from a topic 333
    - jumping to from contents or index 333
  - JavaHelp
    - jumping to 408
    - using a macro for 403
  - OmniHelp, jumping to 370
  - Oracle Help, jumping to 408

# ABCDEFGHIJKLMNOPQRSTUVWXYZ

---

## WinHelp

- jumping to 302
- not jumping to from contents 309

## WinHelp

- advantages and disadvantages of 244
- compiling
  - for delivery 802
- contents levels, setting 250
- contents, configuring 306
- files, identifying 306
- generating 281
  - cross references 288
  - footnotes 288
  - pop-ups from table cells 292
  - tables 290
  - topics from table rows 292
- index entries, maximum length of 252
- macros, invoking 302
- overview topic, renaming or eliminating 309
- platform-specific settings 283
- project file, naming 284
- titles, identifying 306
- topics
  - creating 294
  - from table rows 292
- using for HTML Help pop-ups 323

WinHelp 2000, producing via WinHelp 4 244

WinMerge, file comparison tool 35

WinZip add-on for archiving deliverables 803

## WMF graphics

- limitations of 746

## Word

- stylesheet, generated from output formats 109
- template, specifying 223
- version 2000, converting to 237
- version 8, configuring for 224
- versions of, adjusting for 224

## wrap

- and ship conversion output 787

wrap directory, *see* [assembly directory](#)

\_wrap, default assembly directory 247, 788

## wzzip.exe

- for archiving deliverables 803
- for packaging Eclipse Help topic files 427

# X

## XHTML

- declaration, suppressing 438
- DocType and DTD 433
- encoding, specifying 434
- for Confluence 4.x, generating 442
- OmniHelp viewer files 380
- tagging for HTML output 431
- using instead of HTML 430

XInclude, Calabash 178

## XML

- catalogs, connecting to 67
- catalogs, for specializations 68
- comments, inserting with markers 724
- content type, specifying 451
- editor oXygen, integrating **DITA2Go** with 36
- file extension, specifying 431, 450
- line breaks in, suppressing 439, 451
- links, managing 454
- output settings, specifying 450
- structure, providing 451
- tag names, deriving from CSS 452
- tags, providing 451
- version, specifying 450
- within HTML 566

## .xml

- default XML file extension 431

**XrefBranch**, PI marker for cross references to named map branches 171

**Xrefbranch**, PI marker for cross references to named map branches 720

# Y

Y: *No entries*

# Z

ZIP command for Eclipse Help 427