Copyright and Trademarks

WebMaker User Guide

Version 3.0

April 1997

Part number: WM30UG

Copyright © 1995-1997 by The Harlequin Group Limited.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of The Harlequin Group Limited.

The information in this publication is provided for information only, is subject to change without notice, and should not be construed as a commitment by Harlequin Limited, Harlequin Incorporated, Harlequin Australia Pty. Limited, or The Harlequin Group Limited. The Harlequin Group Limited assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication. The software described in this book is furnished under license and may only be used or copied in accordance with the terms of that license.

WebMaker is a trademark of The Harlequin Group Limited.

"Digitool, Inc. ("Digitool") and its licensor make no warranties, expressed or implied, including without limitation the implied warranties of merchantability and fitness for a particular purpose, regarding MCL. Digitool and its licensor do not warrant, guarantee or make any representations regarding the use or the results of use of MCL in terms of its correctness, accuracy, reliability, currentness or otherwise. The entire risk as to the results and performance of MCL is assumed by you. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you.

In no event will Digitool, its licensor, their directors, officers, employees or agents be liable to you for any consequential, incidental, or indirect damages (including damages for loss of business profits, business interruption, loss of business information, and the like) arising out of the use or inability to use MCL even if Digitool and/or its licensor have been advised of the possibility of such damages.

Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you. Digitool's and its licensor's liability to you for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, tort (including negligence), product liability or otherwise) will be limited to $50."

Other brand or product names are the registered trademarks or trademarks of their respective holders.

The WebMaker Software is subject to the following Restricted Rights Legend: "Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in (i) FAR 52.227-14 Alt III, (ii) FAR 52.227-19, (iii) DFAR 252.227-7013(c)(ii), or (iv) the accompanying license agreement, as applicable. For purposes of the FAR, the Software shall be deemed to be 'unpublished' and licensed with disclosure prohibitions, rights reserved under copyright laws of the United States. Harlequin Incorporated, One Cambridge Center, Cambridge, Massachusetts 02142."

Europe:

Harlequin Limited
Barrington Hall
Barrington
Cambridge CB2 5RG
U.K.

telephone +44 1223 873 800
fax +44 1223 872 519

North America:

Harlequin Incorporated
One Cambridge Center
Cambridge, MA 02142
U.S.A.

telephone +1 617 374 2400
fax +1 617 252 6505

Electronic Access:

http://www.harlequin.co.uk/
http://www.harlequin.com/

Table of contents 36
Index 40
Including external links 41
Special characters 41
Troubleshooting RapidRules output 42

4 Complete WebMaker Options 45

WebMaker Wizard 46
Main window 46
RapidRules dialog 49
Complete options for making a web 50

5 Cascading Style Sheets 55

Overview of CSS 55
How WebMaker supports CSS 56
Hand-crafting the CSS file 57

6 Making a Web in Batch Mode 59

Syntax for running WebMaker interactively 59
Syntax for batch mode 60
Examples of batch mode syntax 64

7 FrameMaker Style Recommendations 65

Use the paragraph and character catalogues 66
Put graphics in anchored frames 66
Attach anchored frames to their own paragraphs 67
Use cross references 67
Use autonumbering 68
Use heading levels consistently 69
Check information within master and reference pages 69
Check the order of text flows 69
Consider imported graphics resolution 70
Consider cross-reference formats 70

8 Troubleshooting 73

The log window 73
Debugging markers 73

Index 157

and a variety of UNIX platforms. The writer has a WYSIWYG interface and wide latitude in page design. The style of the design is specified in paragraph and character tags that define the appearance of the text of the book.

FrameMaker describes the appearance of a document rather than its logical structure. FrameMaker documents are a sequence of independent paragraphs with a flat structure. The list and environment concepts of HTML documents are not really part of FrameMaker. An impression of paragraph hierarchy is achieved only by the formatting of individual paragraphs; that is, by what a paragraph looks like when it is printed on paper.

The FrameMaker paragraph and character catalogs are a collection of formats and not of logical types.

For example, a paragraph tagged with the "Chapter" paragraph format starts and ends with itself. There is no knowledge of the previous or next paragraphs and it is hierarchically at the same level as, and independent of, any other paragraph in the document. However, there is normally the sense that such a paragraph is more important than neighboring ones. This is achieved by the appearance of the paragraph: perhaps large, bold type on a new page.

1.2 WWW documents

WWW documents are written in HTML, the Hypertext Markup Language. HTML describes the hierarchical structure of a document, leaving the presentation of the document to external browsers. Paragraphs may be at different hierarchical levels and may be interpreted differently depending on their environment.

Documents are divided into two elements, marked by the HTML tags of **`HEAD`** and **`BODY`**. The **`HEAD`** contains information about the document itself. The **`BODY`** is the text of the document. It is made up of a sequence of HTML constructs, which may be described by the following three cases:

- Simple paragraphs containing textual and image data with character highlights, hypertext links, and line breaks.

  The basic paragraph, tagged **`P`**; all headings, tagged **`H1`**, **`H2`**,..., **`H6`**; and the preformatted environment, tagged **`PRE`**.

- Environments that contain simple paragraphs.

WebMaker was initially designed by CERN, creators of the WWW itself, to provide a tool for the rapid exchange of scientific information. WebMaker takes a FrameMaker document or book, translates each paragraph or character tag to a set of HTML commands according to paragraph translation rules, and creates a web of WWW documents.

WebMaker is configurable. You can modify existing rules or create new rules that define the HTML translation for each FrameMaker tag. You can save these translation rules to use when you have changed the FrameMaker document source or to use on other documents of the same design. To tune the WWW display, you change these conversion rules, just as you change FrameMaker tags to tune the printed version of a document.

The WebMaker conversion rules use WML, the WebMaker Language. WebMaker includes libraries of translation rules that cover common styles and serve as examples of the use of WML.

1.4 Sample output of WebMaker

The document you are reading, *WebMaker User Guide*, is a good example of a FrameMaker document that is available both in hardcopy and on the WWW. In the following sections, we show some of the pages of this document as they appear in a Web browser.

Note that we control the appearance of these pages by using the library of translation rules, and by specifying the rule we wanted for each FrameMaker tag in our document. You could start with the same FrameMaker document and achieve a very different appearance of the web document.

1.4.1 A web document in one or more HTML pages

One of the first choices you make is whether you want the FrameMaker document to be converted to a single HTML page, or to many HTML pages. For a short document, converting it to a single HTML page may be best. For long documents, it is useful to divide the information into many HTML pages. WebMaker enables you to specify the level of granularity of your document. For example, consider the outline of the *WebMaker User Guide* shown in Figure 1.2.

Figure 1.3 First page of the *WebMaker User Guide*

As you can see in Figure 1.3, the first page of the *WebMaker User Guide* contains three navigational buttons, labeled **Next**, **Contents**, and **Index**. If you click on the **Next** button, you see the "next page" of the web document. If you continue to press **Next**, you can see all the text of the web document in the same order that it appears in FrameMaker, or in the hardcopy document. If you click on the **Contents** button, you go directly to a page that contains a detailed table of contents, as shown in Section 1.4.3, "Table of contents". If you click on the **Index** button, you go directly to a page that contains an index of this document, as shown in Section 1.4.4, "Index".

The navigation buttons appear both at the top and bottom of the page, as a convenience for the people who read this web document. It is always useful to

before this page of the document. In this case, if you click on **Previous**, you go to section 1.3. The **Previous** button is the opposite of the **Next** button.

The **Up** button brings you up one level in the document to the section that contains this subsection. In this case, 1.4 is a subsection of section 1, so clicking the **Up** button in 1.4 brings you up one level to section 1. The first line of text in this page shows the name of the section that includes this section; this line tells you which section is **Up**.

The **Top** button brings you to the top page, or first page, of this web document. In this case, **Top** brings you to the page shown in Figure 1.3.

Note that each page automatically shows only the navigational buttons that are appropriate for that page. Thus, the first page of the web document has no **Top** button (because that page is the top), no **Previous** button (because there is no page previous to the first page), and no **Up** button (because there is no section that is up a level from the first page).

1.4.3 Table of contents

Figure 1.5 shows a portion of the table of contents of the *WebMaker User Guide*, which is the page you see if you click on the **Contents** navigational button.

Figure 1.6 Index of the *WebMaker User Guide*

You can choose whether or not you want your web document to have an index. An index provides another way for readers to navigate your document. WebMaker constructs the index automatically, based on the Index markers in your FrameMaker document.

1.4.5 Cross references and hypertext markers

Webmaker automatically converts FrameMaker cross references to links in the web document. You can click on these links to follow the cross reference; that is, to go to the section that is cross referenced. Thus, all the navigational support you have built in to your FrameMaker document is preserved in the web document.

- Reusable rules files, which enable you to quickly convert other documents which use the same FrameMaker template.
- Ability to define new rules, customized for your documents.
- Ability to run in batch mode on UNIX and Windows.
- Ability to run WebMaker interactively on UNIX, Windows, and the Macintosh.

1.6 Harlequin and WebMaker

The original WebMaker was created at CERN, the European Center for Particle Physics. The initial version of WebMaker was a command-line program that ran only on UNIX. CERN sold the rights to WebMaker to Harlequin.

Harlequin developed a graphical user interface for WebMaker and ported it to the Windows and Macintosh platforms. Harlequin also provides customer support for WebMaker by electronic mail, at:

```
webmaker-support@harlequin.com
```

Harlequin is a global company with offices in the United States (Cambridge, Menlo Park, Seattle), the United Kingdom (Cambridge, Manchester, Edinburgh), and Australia (Sydney, Canberra, and Brisbane).

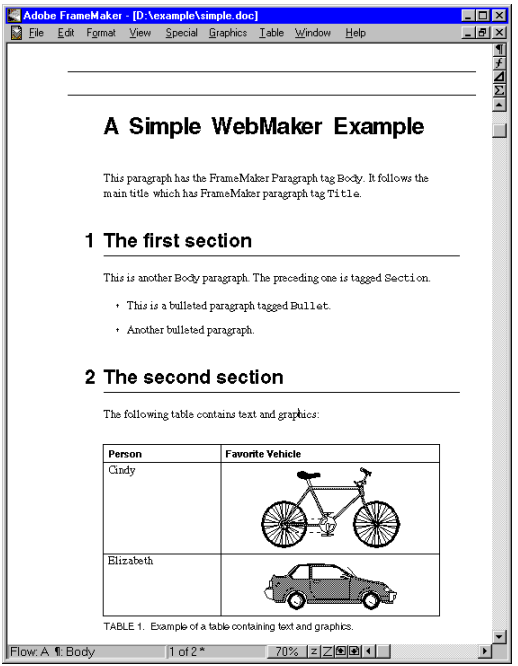

Figure 2.1 FrameMaker file: **`simple.doc`**

2.2 Write the FrameMaker file to MIF format

Before running WebMaker, you need to save the FrameMaker document in MIF format, to a file with an extension of **`.MIF`**. Note that the file extension must be **`.MIF`**, and cannot be **`.mif`**.

To save a file in MIF format:

Figure 2.2 WebMaker Wizard

The WebMaker Wizard enables you to customize the user interface based on your level of experience with WebMaker. Here, select **New User** and click **Do It**.

2.4 Use RapidRules

WebMaker provides a tool called RapidRules which makes an initial guess at the correct WML rules for your FrameMaker document. The guesses will not be perfect; some adjustment of WML rules and mappings may be necessary to achieve the results you want for a final conversion.

To use RapidRules, follow these steps in the Wizard:

1. Click **Use RapidRules** and then click **Do It**.

2.5 Make the web document

You can make a web document by clicking **Make a Web** in the main window, or **Convert MIF File to HTML** in the Wizard. Both those commands open the Make a Web dialog, as shown in Figure 2.4.

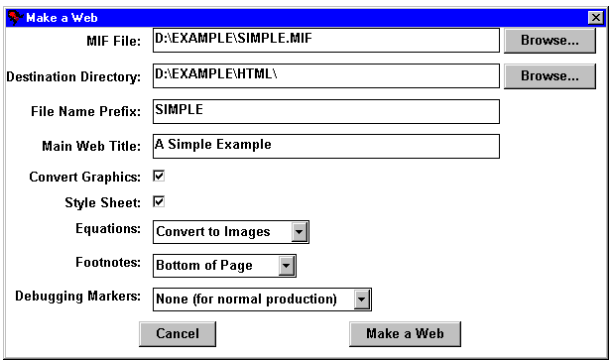

Figure 2.4 Make a Web dialog

You need to specify the following items:

- The FrameMaker document: this appears by default as the same document you specified for RapidRules, which is **`simple.MIF`**.
- The destination directory: this is the directory where WebMaker will create the HTML files.
- The Main Web Title: this is the title of the web document you will create. Type in “A Simple Example”.
- Click **Make a Web**. The progress bar in the main window or the Wizard will show WebMaker read in the MIF file and generate the HTML.

For more information about the Make a Web dialog, see Section 4.4, “Complete options for making a web”.

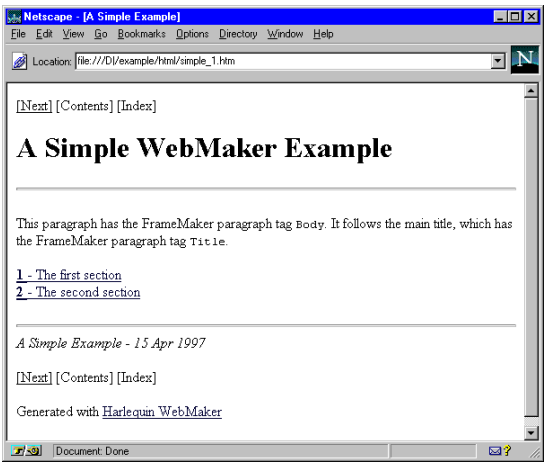

Figure 2.5 First web page of A Simple Example

The Document Title is “A Simple Example”; this was the title we provided in the Make a Web dialog. Notice that this page has a **Next** button at the top of the page and another one at the bottom, for easy navigation. This page contains some text, and two links. You can go to the next section by clicking on the **Next** button, or by clicking on the first link.

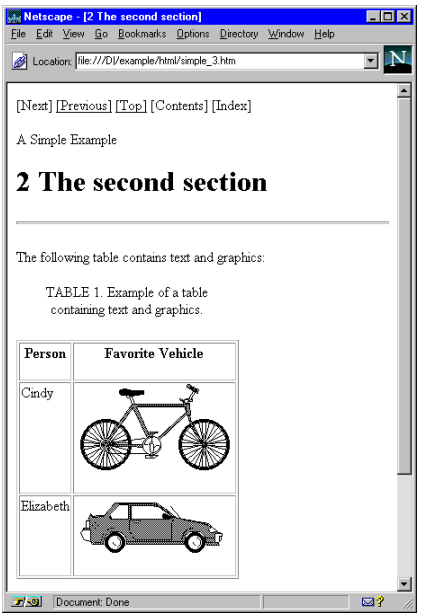

Figure 2.7 Third web page of A Simple Example

The third page has three navigation buttons (which appear at the top and at the bottom of the page). **Next** is grayed out and does not go anywhere, because this is the last page; **Previous** goes to the previous page; and **Top** goes to the first page of this web document.

If you are interested in seeing the HTML itself, you can give the command in your browser to view the HTML source of any of these web pages.

4. If your WML file includes the **`nodesB.wml`**, **`nodesBTI.wml`**, or **`node-java.wml`** library, your HTML files will have graphical navigation buttons. You must copy the **`.gif`** files containing the graphical buttons from the lib directory to the directory containing the HTML. See Section 3.5.2, “Graphical navigation buttons”.
5. If the table of contents needs adjustment, see Section 3.6, “Table of contents”.
6. If you want to include an index, see Section 3.7, “Index”.
7. If you want to include a list of figures or list of tables, see Section 12.10, “List of figures or tables in WML”.
8. If you want to include links to URLs outside this document, see Section 3.8, “Including external links”.
9. For frequently asked questions about RapidRules output, see Section 3.10, “Troubleshooting RapidRules output”.
10. Consider the quality of the FrameMaker source document. We offer style recommendations for making documents convert cleanly. See Chapter 7, “FrameMaker Style Recommendations”.
11. If you find that you need a paragraph rule that is not provided in the WebMaker libraries, you can define your own rule, using the WML language. See Chapter 10, “Modifying Conversion Rules, Headers, and Footers”.
12. You can change the navigation buttons and the text appearing at the top of each web page (its header) and the bottom of each web page (its footer). See Chapter 10, “Modifying Conversion Rules, Headers, and Footers”.
13. Another way to control the appearance of the web pages is to customize the cascading style sheet produced by WebMaker. See Chapter 5, “Cascading Style Sheets”.

found FrameMaker document elements to WML paragraph and character rules. This section is particularly handy when you have a WML file generated by RapidRules and you wish to make some changes to the mappings it chose. For a complete description of each WML rule, see Chapter 11, "WML Library Rules Reference".

3.2.1 Structuring converted webs

You can think of a WebMaker *node* as an individual web page. In WebMaker, you can generate webs made up of one or many nodes. A *single–node web* consists of a single HTML file; a *multiple–node web* has many HTML files. WebMaker can convert a FrameMaker document or book into either type of web.

The WML file for a multiple–node web generates nodes by mapping FrameMaker heading tags to node-generating WML rules. The WML file for a single–node web maps FrameMaker heading tags to plain HTML heading tags. For single–node webs, you can choose WML heading rules that will format the text to visually differentiate between logical levels. See Section 11, "WML Library Rules Reference" for more information on the available rules.

We recommend that you choose multiple–node webs for any document more than two or three pages in length. You should use a single–node web only if your document is small and simply structured. We recommend this for two reasons:

- A single–node web results in one HTML file, which for large documents is difficult to navigate.
- A single–node web does not allow you to take full advantage of WebMaker's ability to implement structure and logical organization.

WebMaker works best on documents with a consistent hierarchical structure, such as technical documents with numbered sections. WebMaker creates hypertext links based on that organization as well as on standard cross references. Inconsistent heading levels (for example, level 1 followed directly by before level 3) or "orphan" headings (sections with only one subsection) do not work as well as consistent structure in either FrameMaker or WebMaker. You can use FrameMaker's generated lists (such as a table of contents) to troubleshoot your document's structure.

Table 3.1 Paragraph format mappings

Paragraph type	WML rule	
	Multiple Nodes	Single Nodes
headers of a page	Ignore	same
heading: top level	L1H1-NodeTop	L1H1
heading: second level	L2H1-NodeLower	L2H2
heading: third level	L3H1-NodeLower	L3H3
index formats (such as Level1IX)	Ignore	same
numbered list: first level	L1AutonumberItem	same
numbered list text: first level (non-numbered text indented at same level as numbered list)	L1PlainItem	same
numbered list: second level	L2AutonumberItem	same
numbered list text: second level (non-numbered text indented at same level as numbered list)	L2PlainItem	same
numbered list: third level	L3AutonumberItem	same
numbered list text: third level (non-numbered text indented at same level as numbered list)	L3PlainItem	same
plain text	Default	same
preformatted text	PreformattedText	same

You can map a character tag to None to give those characters no HTML highlight.

3.3 How to use format override rules

We recommend that you use character tags in FrameMaker whenever you want to specify the appearance of characters. (See Section 7.1, "Use the paragraph and character catalogues".) However, some FrameMaker documents do contain characters that have no character tag, but do have formatting applied to them. You can use WebMaker to specify how such characters appear when converted to HTML.

To specify how WebMaker converts characters that use format overrides, you need to define a *format override rule* that maps the character style to the HTML highlight name.

WebMaker recognizes the **`ANGLE`**, **`FAMILY`**, **`VAR`** and **`WEIGHT`** parameters of FrameMaker characters. Format override rules may specify different combinations of these parameters to match characters with no tags but with formatting applied to them and map them to HTML highlights. You can map a character tag to **`None`** to give those characters no HTML highlight.

For example, you might create format override rules such as these:

- For characters whose angle is oblique, map them to the Italic HTML highlight.
- For characters whose angle is oblique and whose family is Courier, map them to the Variable HTML highlight.

Note that characters can match more than one format override rule. For example, consider characters whose angle is Oblique and family is Courier. Those characters match both the rules listed above.

In cases where characters match more than one format override rule, the most detailed rule takes precedence. In this example, the second rule is more detailed than the first rule because it specifies more aspects of the characters. Thus, the second rule is used for characters that match both rules.

You can create format override rules using WebMaker's graphical user interface or by writing WML rules directly (see Section 12.8.2, "Syntax of format override rules").

3.4.1 Choosing which WML files to include

This section describes how to choose the appropriate WML files from the WebMaker library.

Generally, you always want to include these files:

`headings.wml` Contains paragraph rules for headings.

`normals.wml` Contains paragraph rules useful for body text, including an Ignore rule.

`lists.wml` Contains paragraph rules for lists, bullets, numbered lists, and glossaries.

If your document will not contain more than one node, you should not include a nodes WML file.

If your document will contain more than one node, you should choose exactly one of these nodes WML files:

`nodes.wml` You should use this file if you use headings that create new nodes, and if you do not want to use images for navigation buttons or create a table of contents or index.

`nodesTI.wml` You should use this file if you use headings that create new nodes, and if you want to create a table of contents and index. This file uses text rather than images for navigation links.

`nodesB.wml` You should use this file if you use headings that create new nodes, and if you do not want to create a table of contents or index. This uses the button GIF files for images for the navigation buttons. If you include this file, you need to copy the button GIF files from the **`lib`** directory to the directory where the HTML files are stored. See Section 3.5.2, “Graphical navigation buttons”.

3.5 Graphics and tables

A web created through WebMaker is a set of HTML files and graphic files in Graphic Interchange Format (GIF) format. When you view the web, the browser displays the graphics as inline images in the appropriate positions within the HTML text of the web document.

WebMaker converts FrameMaker tables into HTML 3.0 tables. The conversion of tables is automatic and requires no configuration choices.

3.5.1 Converted graphics

WebMaker converts anchored frames to GIF files, a format that most browsers can display. It writes these GIF files to the same directory as the HTML files.

WebMaker converts into GIF both graphics copied into the FrameMaker document and graphics imported by reference. Graphics imported by reference must, of course, be available in the location specified in FrameMaker. WebMaker ignores any graphics outside anchored frames.

On UNIX systems, WebMaker uses the **`fmbatch`** utility provided as part of FrameMaker for graphic conversions. Make sure **`fmbatch`** is in your path.

3.5.2 Graphical navigation buttons

If you use the **`nodesB.wml`**, **`nodesBTI.wml`**, or **`nodejava.wml`** library, the rules in those libraries specify graphical navigation buttons that appear in the header and footer of each HTML page. That is, each HTML page includes links to files named **`next.gif`**, **`prev.gif`**, **`top.gif`**, and so on. See Section 11.2, “Node libraries” for a complete list of the GIF files.

WebMaker provides graphical buttons in the **`lib`** directory. You need to copy all the **`.gif`** files from the lib directory to the directory containing your HTML files.

3.6 Table of contents

WebMaker can create three different kinds of tables of contents in the web document. A table of contents in the web document contains links to the chapters and sections rather than supplying page numbers.

- Map the tags from the FrameMaker table of contents to TOC-Entry rules. (If you use RapidRules, it will map these tags appropriately for you.) For example, if your table of contents contains entries for Chapter and 1Heading, the paragraph tags are ChapterTOC and 1HeadingTOC. You could map the ChapterTOC paragraph tag to the TOC-Entry-1 rule and the 1HeadingTOC paragraph rule to the TOC-Entry-2 rule. These TOC-Entry rules vary only in the level of indentation.

Note: the TOC-Entry rules are defined to work for a table of contents of the following format: each entry contains a number (optional), a title, some white space (spaces or tabs), and a page number. The rules strip the page number from the mapped table of contents. If your table of contents has a different format, you may need to define customized versions of the TOC-Entry rules. For general information about editing WML, see Section 10.1, “Creating new WML conversion rules”.

Using a similar technique, you can convert similar kinds of lists generated by FrameMaker, such as a list of tables, list of figures, and so on. To do so, you must have generated these lists with the Create Hypertext Links box checked. The **`contents.wml`** file contains rules for converting these kinds of lists. For more information, see Section 11.5.1, “Table of contents and related rules”, and Section 12.10, “List of figures or tables in WML”.

3.6.3 Generated table of contents

A *generated table of contents* is a node that contains a table of contents for the whole web of the FrameMaker document. A generated table of contents contains entries for each tag that you have mapped to headings level 1 through 4. WebMaker generates the table of contents from the paragraph tags in the document files, and does not use the table of contents in your FrameMaker document (if any). Thus, you can create a generated table of contents even if your FrameMaker document does not have a table of contents.

The **`nodesTI.wml`** and **`nodesBTI.wml`** library files contain a rule that creates a generated table of contents.

Both local and generated WebMaker tables of contents use paragraph tags mapped to Headings rules for the tables. Paragraphs whose tags are mapped to Headings rules become entries in the web’s tables of contents. These tables

If RapidRules finds no paragraph tag named WWW-TOC but does find paragraph tags with names ending in TOC (such as ChapterTOC and 1HeadingTOC), it creates a mapped table of contents. It includes the **contents.wml** file in the WML file it creates. It maps all paragraph tags whose names end in TOC to the TOC-Entry rule. It causes the Contents navigation button to link to the first HTML page containing a table of contents entry.

3.7 Index

WebMaker can create an index to the web document; this is a generated index (much like a generated table of contents), not a mapped index. WebMaker creates the index based on the index markers in the FrameMaker document. The index entries are links in the web document and do not show page numbers.

To have WebMaker create an index:

- Create a paragraph tagged with a special tag, such as WWW-IX, once and only once in the FrameMaker document. This can be the title of the index, but it must be a unique tag rather than an instance of Chapter or 1Heading.

Include a library file such as **nodesTI.wml** and **nodesBTI.wml** in your WML file. These library files define node rules that create an index.

- Include the **headings.wml** library file in your WML file. This file defines paragraph rules that use index node rules.
- Map the index tag to a paragraph rule that uses an index node rule, such as **IndexHeadingS-IndexNodeS**, in your WML file.

WebMaker offers three styles of index:

Simple index	Creates a single node. Index entries are sorted alphabetically and displayed without headers dividing the entries into sections. The **IndexHeadingS-IndexNodeS** rule in **headings.wml** produces this.
Letter index	Creates a single node. Index entries are sorted alphabetically and displayed with headers dividing the entries into sections for each letter. The **IndexHeadingL-IndexNodeL** rule in **headings.wml** produces this.

serif trademark symbol, ™, has no encoding in HTML 3.2. WebMaker converts these special characters to appropriate substitutions. For example, the trademark symbol is converted to the letters TM in superscript.

Note that older versions of browsers do not properly display all the special characters. If your readers see the encodings appear in the browser window instead of the special characters, they should obtain a more current version of the browser.

Note also that characters in symbolic fonts, such as Zapf Dingbats and Symbol, do not have HTML representations, and do not have appropriate substitutions.

3.10 Troubleshooting RapidRules output

When you inspect the HTML output generated using RapidRules's WML file, you may find that some of the results are not quite to your liking. This section presents some common questions and answers about how to change the WML to achieve the results you want.

- Why is the Index link at the top of each web page greyed out?

  RapidRules maps FrameMaker paragraph style tags named WWW-TOC and WWW-IX to WML rules that create a generated table of contents and an index for your web. If you do not have a tag in your FrameMaker document mapped to these WML rules, WebMaker cannot create an index for you. See Section 3.6, "Table of contents", for information on how to tell WebMaker to make a table of contents for you. See Section 3.7, "Index", for information on how to tell WebMaker to make an index for you.

- Why do I have an Index page that has only a title and no entries?

  Even if you map some FrameMaker tag to an index-node-generating WML rule (like IndexHeadingL-IndexNodeL), WebMaker cannot create an index if your FrameMaker document does not have any index markers. If you want a web index, you must have index markers in your FrameMaker document.

- Why don't the navigation panels at the top of each web page have graphic buttons?

verted. Because the anchored frame surrounding the graphic is then too small, the image will be cropped. For best conversion results, you should make sure that the images you want to convert are imported at 72 dpi in your FrameMaker document.

See Chapter 7, "FrameMaker Style Recommendations" for more information about anchored frames and graphics conversion.

- Why are my nodes beginning at incorrect places in my document?

  Some kinds of headings are difficult to distinguish from other headings. Therefore, RapidRules might map both a chapter name and a chapter number to node-generating WML rules, resulting in two nodes, when you only want one node. It might also map only subheadings to nodes, instead of chapter or section headings. You should browse the HTML files in more than one browser to see if you can determine the kind of FrameMaker paragraph that is getting an incorrect mapping. (Turning on the Debugging Markers option in the Make a Web dialog box can make this process somewhat easier; see Section 8.2, "Debugging markers".) Then, you can modify your WML file to use the correct rules.

  RapidRules may also wrongly identify some headings as not being headings. For instance, this document's copyright page logically belongs in its own web node, but the text's formatting is not obviously that of a heading; RapidRules thus thinks that it is not a heading. You should edit the WML file to make such a heading map to a node-generating WML rule.

  An unusual autonumber sequence in heading definitions may also cause nodes to appear at seemingly odd places. If your system of autonumbering is not completely straightforward, the headings may be mapped to inappropriate WML rules; for instance, a top-level heading might appear smaller than a second-level heading. You should edit the WML file to reflect the true organizational structure of your document.

4.1 WebMaker Wizard

The WebMaker Wizard enables you to customize the user interface based on your level of experience with WebMaker. You can choose one of three levels.

Table 4.1 Levels of experience in Wizard

Level	Behavior
New User	Maximum amount of help is available in Wizard. No Close button is available for closing the Wizard.
Somewhat Experienced User	Same as the New User option except that the Close button is available for closing the Wizard.
Very Experienced User	Wizard is closed and you use the main window.

You can accomplish all the same tasks regardless of which level you choose. If you like using the Wizard to lead you through the steps, choose New or Somewhat Experienced User. Once you are comfortable with the main window, you will probably find it faster to work without the Wizard, by choosing Very Experienced User.

4.2 Main window

The main window is shown in Figure 4.1.

- **Stop** interrupts a running process such as RapidRules or making a web.

4.2.2 Menu commands

The **File** menu offers commands that operate on WML files:

- **File > New** creates a new, empty WML file.
- **File > Open** opens an existing WML file.
- **File > Close** closes the current WML file.
- **File > Save** saves the current WML file.
- **File > Save As** saves the current WML file, offering you the opportunity to specify a different filename.
- **File > Use RapidRules** opens the RapidRules dialog
- **File > Quit** exits WebMaker.

The **Edit** menu offers commands that alter the current WML file:

- **Edit > Import Frame Tags** enables you to read in a MIF file (single document or book) and create new paragraph and character rules for tags that appear in that document but do not yet exist in the WML file. This command is useful when you have created a WML file that matches a certain document, and you want to use it on another document, but you suspect that the second document uses additional FrameMaker tags. When the new tags are added to the WML file, they are not mapped to conversion rules; you must do that step yourself.
- **Edit > Add** adds a new WML library (in the WML Libraries view) or adds a new format override rule in the (Format Overrides view).

The **WebMaker** menu offers these commands:

- **WebMaker > Make a Web** opens the Make a Web dialog so you can convert a document.
- **WebMaker > Log** opens the log window, which displays detailed information about the conversion process, including any errors encountered.

The **Help** menu offers these commands:

When you have filled in the filename and options, click **Generate Rules**. RapidRules then creates a WML file based on the MIF file you specified and displays it in the WML window.

RapidRules names the newly generated WML file based on the name of the MIF file. For example, if you specify **`user-guide.MIF`**, the WML file will be named **`user-guide.wml`**. In this case, the WML is named **`simple.wml`**.

You can use RapidRules repeatedly, which is useful if you are making changes to your FrameMaker file that affect the Paragraph or Character Catalogs.

At this point, the generated WML file is not yet saved. You can save the WML file at any time by using **File > Save** or **File > Save As** (if you wish to change the name of the file).

4.4 Complete options for making a web

First, if you want to see detailed information about what WebMaker is doing as it makes the web, choose **WebMaker > Log**. That command opens a window containing a log of status information.

To make a web, either click **Make a Web** or choose **WebMaker > Make a Web**. A dialog appears, enabling you to specify options about the web. The dialog is shown in Figure 4.3.

length limitation, if that limit is reached (by means of a file such as **sim-10000.htm**), WebMaker continues to number the files but it removes a character from the prefix in order to adhere to the filename length limitation (for instance, **si-10000.htm**).

Now you should fill in the other options:

1. Enter a Main Web Title. This is a title that should describe the entire web document; it is often the same as the title of your FrameMaker document. There is no default.
2. If you are using UNIX or Macintosh, choose a Filename Format option. The default value is Long File Names.

Table 4.2 Filename Format Options

Filename Format Option	Behavior
Long File Names	Uses long, descriptive names for the generated HTML and image files (for example, **simple-1-image-1.gif** for the first image file on the first page of the web).
8.3 Lowercase File Names	Uses filenames that conform to the DOS 8.3 file naming conventions, and generates files that have lowercase names (for example, **sim1ima1.gif** for the first image on the first page of the web).
8.3 Uppercase File Names	This option also uses filenames that conform to the DOS 8.3 file naming conventions, but it uses uppercase file names (for example, **SIM1IMA1.GIF** for the first image on the first page of the web).

3. Choose the Convert Graphics option. If checked, all graphics in the document are converted to GIF. If unchecked, the graphics are not converted, but the links to them are created in the HTML files; this option can save you time if you are reconverting a document, the graphics have

referenced external style sheets. In addition to the style sheets provided by the author, the browser can provide a style sheet (to be used as the default), and the reader can provide a style sheet. The term *cascading* is intended to convey the process of combining all the style sheets in effect. The CSS specification defines the rules for combining style sheets.

5.2 How WebMaker supports CSS

When you make a web, you have the choice of generating a style sheet or not generating it. If you choose to generate a style sheet, these things happen:

- WebMaker creates a style sheet in a **`.css`** file that contains style commands that mimic the formatting in the FrameMaker document. In the style sheet, each paragraph tag and character tag has an entry that specifies its formatting. If you convert a book file, one **`.css`** file is created for the book. The name of the **`.css`** file is the same as the name of the HTML files, but with the **`.css`** extension. Thus, if you convert a document named **`guide`** and do not specify a different prefix, WebMaker creates HTML files named **`guide-1.html`** and so on, and a style sheet named **`guide.css`**.
- Each HTML file references the **`.css`** file.
- The HTML elements have a **`class`** attribute. By default, the class name is the same as the name of the FrameMaker tag.

If you do not generate a style sheet, no **`.css`** file is created, the HTML files do not reference any **`.css`** file, and the HTML elements do not have a **`class`** attribute. The browser will control the formatting of the HTML files.

5.2.1 FrameMaker formatting supported by WebMaker CSS

When WebMaker creates the **`.css`** file, it tries to preserve the following formatting information in the FrameMaker file: font, size, color, space above, space below, indentation, first-line indentation, weight, angle, line spacing, and alignment.

It is important to remember that whenever you convert the FrameMaker document using the **Style Sheet** option, WebMaker will recreate the `.css` file. Thus, if you have edited that `.css` file, you must copy it to another filename before converting the FrameMaker document, and then copy your `.css` file back to the filename that WebMaker expects.

5.3.1 Batch mode options for CSS

You can use the batch mode conversion to control more aspects of how WebMaker handles the CSS. You can specify the filename of the `.css` file that WebMaker creates. You can also cause WebMaker to write HTML files that use a `.css` file of another name. By using those options together, you can have WebMaker create a style sheet named `mydoc.css`, but have the HTML files use a style sheet named `custom.doc`. You can edit `custom.doc` to use `mydoc.css` but override certain formatting in `mydoc.css`.

See Section 6.2.3, “Options for style sheets”.

6.2 Syntax for batch mode

To run WebMaker in batch mode:

webmaker [*batch-options***]** ***document***

The complete set of *batch-options* is:

```
[-r | -R WML-file | -RR WML-file | -c WML-file]
[-single]
[-styles CSS-file | -nostyles] [-linkstyle name]
[-t title] [-w directory] [-log filename] [-n name]
[-f b|o|i] [-e h|c] [-8 u|lower|long] [-nographics]
[-limit #pages] [-debug full|summary]
```

For batch conversion, you must supply the *document* argument:

document	A file in FrameMaker's MIF format. It can be a single document file or a book file. The document must have a **.MIF** suffix. In the case of a book, specify only the MIF of the book itself; note that the MIF of each component file must be in the same directory and have a **.MIF** suffix.

6.2.1 Options for RapidRules or existing WML conversion

You must specify exactly one of these *batch-options*:

```
[-r] | [-R WML-file] | [-RR WML-file] | [-c WML-file]
```

-c ***WML-file***	Converts the *document* according to WML rules contained in the *WML-file*.
-r	Run RapidRules and use the resulting WML to convert the document. Save the WML in a file whose name is based on the document filename, but with a **.wml** suffix. The file **document.wml** must not exist.
-R ***WML-file***	Behaves like the **-r** option, but writes the WML to the *WML-file* you specify.

-limit *#pages* Limit the conversion to the number of HTML pages specified by *#pages*. This option is useful for debugging; if you are interested in seeing something on the third HTML page, you do not need to convert the remaining pages.

-nographics Suppress the conversion of graphics.

-debug full Generate full debugging information in HTML output.

-debug summary Generate summary debugging information in HTML output.

-8 u Use uppercase 8.3 filenames for HTML and image files.

-8 lower Use lowercase 8.3 filenames for HTML and image files.

-8 long Use long filenames for HTML and image files.

6.2.3 Options for style sheets

By default, WebMaker generates a Cascading Style Sheet based on the paragraph tags and formatting properties in the FrameMaker document. The default filename of the generated style sheet is based on the Web page file name prefix. WebMaker style sheet files always have a **.css** file type.

When WebMaker generates a style sheet, most HTML elements in the output Web pages are marked with a CLASS attribute that is derived from the corresponding tag in the FrameMaker document. This attribute allows a style sheet to use the CLASS name to control the formatting of Web page elements.

By default, when WebMaker generates a style sheet, a LINK tag is generated in the HEAD of each HTML file that points to the generated style sheet.

You can provide options to control whether to generate a style sheet, its filename, and what style sheet to specify in the LINK tag.

-styles ***CSS-file*** Specifies *CSS-file* as the filename for the generated style sheet.

6.3 Examples of batch mode syntax

To convert a document based on an existing WML file:

```
webmaker -c rules.WML document.MIF
```

To convert a document using an existing WML file, specifying title and output directory:

```
webmaker -c rules.WML -t "My document" -w /pub/mydocs/ mydoc.MIF
```

To convert a document using RapidRules, use one of these command lines:

```
webmaker -r document.MIF                (creates document.WML)
webmaker -R rules.WML document.MIF      (creates rules.WML)
webmaker -RR rules.wml document.MIF     (overwrites rules.WML)
```

7.1 Use the paragraph and character catalogues

The basic principle is that you should create a template containing the paragraph tags and character tags that you need, and you should use the paragraph and character tags consistently. In particular, do not make format overrides to paragraphs or characters. A *format override* is a change that you make to one or more selected paragraphs or characters, without creating a paragraph or character tag that defines the format.

If you use paragraph and character tags consistently, and you do not use format overrides, you gain these advantages:

- You can easily apply a global formatting change to all the paragraphs (or characters) having a given tag.
- You can apply a translating rule to the paragraphs (or characters) having a given tag with WebMaker.

The problem with using format overrides is that you cannot refer logically to the paragraphs that have the format overrides. Thus, although you have carefully selected them and applied formatting to them in the FrameMaker document, you cannot apply a global change to all of them, and you cannot apply a translating rule to them with WebMaker. (WebMaker does provide a facility for handling format overrides to characters, but not to paragraphs. See Section 3.3, “How to use format override rules”.)

If your documents do contain format overrides, you can map them to HTML highlights. See Section 3.1, “Changing mappings”.

Sometimes it is tempting to solve a formatting problem with a quick fix using FrameMaker’s formatting capabilities, but we recommend resisting that temptation. If you use paragraph and character tags consistently, you make it easier to maintain the printed document, and you create a document that is easy to convert to HTML.

7.2 Put graphics in anchored frames

Generally, you want graphics that appear in your FrameMaker document to appear in the converted web document. WebMaker does include the graphics in the converted Web, but only if the graphics are in anchored frames.

cross references to mouse-sensitive hypertext links, but only if you use real cross references in the FrameMaker document.

To create a real cross reference, use FrameMaker's **Special > Cross-References** command. This command creates a logical cross reference, which WebMaker can handle. In contrast, if you simply type in words like: "See Chapter One", the result is plain text, not a logical cross reference. WebMaker has no way of knowing that the words "See Chapter One" should be converted to a link to another web node.

If you use logical cross references, you gain these advantages:

- You can update them automatically in FrameMaker, so that if the name, number, or page number of the referenced section changes, the entire document is updated to the new information.
- You have provided the information that enables WebMaker to easily translate cross references to hypertext links.

Cross references within the same MIF file or book are automatically translated to hypertext links. However, cross references between independent documents are not converted to hypertext links.

7.5 Use autonumbering

For any paragraphs that require a number, you should use FrameMaker's autonumbering facility. For example, you might like headings and some kinds of list items to be numbered.

If you use FrameMaker's autonumbering for paragraphs, you gain these advantages:

- FrameMaker automatically adjusts the numbers when you add or delete numbered paragraphs.
- Your converted web of documents can reflect FrameMaker's powerful autonumbering capabilities. HTML numbering is very limited, providing only a single arabic numeral counter. WebMaker enables you to translate FrameMaker paragraph numbers as text, thus virtually extending the numbering possibilities of HTML to those of FrameMaker.

determined according to the following criteria, in order of decreasing precedence:

1. The text flow that starts on the page with the smallest page number.
2. The text flow having its first text column leftmost on the page.
3. The text flow having its first text column topmost on the page.

If you use WebMaker to convert documents with more than one text flow, your node template should be arranged in such a way that this sorting scheme accurately reflects the logical flow of your document.

7.9 Consider imported graphics resolution

Documents are usually printed at much higher resolution than is available on a computer screen. Graphics are often imported into FrameMaker at 150 or 300 dpi to take advantage of printer resolution and to scale the figure to an appropriate size on the page. But graphics imported at 72 dpi display much better on the screen in WWW documents.

In cases where you want to optimize a document for both the printed page and the screen, you may want to import the same image twice, once at a higher resolution for the printed document and once at a lower resolution for the WWW version of the document. You can then use FrameMaker conditionals to tag the images — using the Paper conditional for the higher-resolution image and the WWW conditional for the lower-resolution one. For information about FrameMaker's conditional feature, see the FrameMaker documentation.

7.10 Consider cross-reference formats

Many FrameMaker documents contain cross-reference formats that mention a page number. For example: See "Consider cross-reference formats" on page 70. The entire cross reference is converted to a link. Because the page number is meaningless in the web document, you might prefer to eliminate it.

If you want to eliminate page numbers in cross references, you can change to a cross-reference format that mentions the section number and name instead of

<<NODE NodeRuleName HEADER>>

<<NODE NodeRuleName FOOTER>>

<<FOOTNOTES>>

<<¶ParagraphName>> *(used for primary paragraph rules)*

<<¶ParagraphName ®ruleName>> *(used for uses paragraph rules)*

The Summary option displays debugging markers only when the WML rules change for the paragraphs; a paragraph that uses the same WML rule as the previous paragraph does not have an additional marker.

8.3 Problems that can occur

These are the kinds of problems that can occur in WebMaker during the conversion process:

- Problems with results due to RapidRules

  Because RapidRules is not perfect, you may encounter some strange effects when your document is converted. See Section 3.10, "Troubleshooting RapidRules output" for some tips on how to resolve some of the problems peculiar to RapidRules.

- Problems due to anomalous FrameMaker usage.

  Sometimes a document looks fine in FrameMaker, but when you convert it to HTML, you find problems with the output. Documents that can be converted smoothly and cleanly are those that follow the suggestions in Chapter 7, "FrameMaker Style Recommendations". You might also find that if you use tricky techniques in FrameMaker, those techniques might not work well when converting to HTML. For example, one document could use autonumbers in some paragraph tags and make the autonumbers invisible in the FrameMaker document. The autonumbers are still present in the document and its MIF file; therefore they are visible in the HTML files generated by WebMaker (that is, if you use the Default rule, or other rules that display the autonumber).

- Problems due to incorrect WML usage.

Library users can use RapidRules in conjunction with the WebMaker library of WML rules to quickly and conveniently convert documents to HTML. They can also compose WML files from scratch using the library WML rules. We believe that most WebMaker users fall into this category. The library users need to understand the following WML elements:

- Include files: see Section 9.2, “Overview of include files”.
- Paragraph rules: see Section 9.3, “Overview of paragraph rules”.
- Character rules: see Section 9.4, “Overview of character rules”.

If, however, you want your document to appear differently than is possible to achieve with the libraries, you can be a *WML configurer*. The WML configurer defines new WML rules. Typically, the WML configurer duplicates a rule from one of the library of WML files, and modifies that rule. If you are a WML configurer, you need a complete understanding of WebMaker Language. In addition to the elements listed above, WML configurers need to understand the rest of the WML elements:

- Mapping rules for format overrides: see Section 3.3, “How to use format override rules”.
- Variables: see Section 9.6, “Overview of variables”.
- Node rules: see Section 9.7, “Overview of node rules”.

The distinction between these two types of users is for the purpose of discussion in this document only; your computer system probably does not distinguish between these two in any way.

9.2 Overview of include files

An *include file* is another WML file that is included in the one you are editing. All the definitions contained in the include file can be used in this WML file.

You can see the include files by choosing the **WML Libraries** view in the main window. For example, on UNIX you might see:

```
/webmaker/lib/nodesTI.wml
/webmaker/lib/headings.wml
/webmaker/lib/normals.wml
/webmaker/lib/contents.wml
/webmaker/lib/lists.wml
```

- A reusable WML file corresponding to the template used in your company.

  The company-specific WML file should include the appropriate WML library files. If you typically use FrameMaker's book facility to organize your documents, the WML template should contain elements that map the paragraph tags and character tags of all the files in the book (which might include the Title page template, a Copyright page template, a TOC template, a Chapter template, an Appendix template, and an Index template).

- A WML file that is specific to a particular document.

  This is necessary only if the document contains extra paragraph tags or character tags that are not in the regular template, or if you want that particular document to have a different appearance in HTML than your other documents.

9.3 Overview of paragraph rules

A WML file can contain definitions of paragraph rules. A paragraph rule specifies the appearance of paragraphs that use that paragraph rule.

9.3.1 Categories of paragraph rules: primary and uses

There are two categories of paragraph rules: primary rules and uses rules. A *primary rule* defines a WML paragraph rule from scratch. The following example is the definition of a primary rule named **`L1BulletItem`**:

```
PARAGRAPH "L1BulletItem" TYPE List
{
  LEVEL 1
  KIND Bullet
  ACTIONS
  {
    write(*,listitem(paragraph(text())));
  }
}
```

A *uses rule* specifies that one paragraph rule uses the definition of another rule. The other rule is typically a primary rule. The following example is the

You can see the format override rules that are defined in a WML file by selecting the **Format Overrides** view in the main window.

9.5 Rules defined more than once

It can happen that a WML element is defined more than once in a WML file. When a WML element is defined more than once, the last definition supersedes all the preceding definitions. Here are some ways this happens:

- A rule is defined once in an include file and once in the WML file itself. The definition in the WML file supersedes the one in the include file. This is the most useful case: it enables you to use an include file, and to modify one or more of the rules that come from that include file.
- A single WML file contains three definitions of the same rule. The third definition supersedes the preceding ones.
- A rule is defined in two files that are included in this WML file. The definition in the last include file supersedes the definition in the preceding include file.
- A WML file includes the same WML file twice; in this case, all rules are repeated. The second set of definitions supersedes the first set (although there is no real effect, because both sets of definitions are the same).

WebMaker detects when a rule is defined more than once, and opens a dialog to inform you.

9.6 Overview of variables

A WML file can contain definitions of variables. A variable in WebMaker is much like a variable in other programming languages. A variable has a name, and you can assign it a value. You can get the value of the variable by using its name.

You can see the variables that are defined in a WML file by viewing the WML file in an ordinary text editor. For example:

```
#VARIABLE @NavPanel
```

Variables are typically used in node rules. To see how the **`@NavPanel`** variable is used, see Section 9.7, “Overview of node rules”.

```
PARAGRAPH "ExtTOCHeading-TocNode" TYPE Heading
{
  LEVEL 1
  FILENAMEKEY TOC
  NEWNODE TOCNode
  ACTIONS
  {
    write(*,heading(1,text()));
    write(*,hrule());
  }
}
```

1. Identify the predefined conversion rule that most closely matches the conversion behavior you want to achieve. In this example, we choose the **QuotedText** rule. It indents text they way we like, but we want to make that text appear in italics.
2. Use an ordinary text editor to edit a new file that contains a copy of that conversion rule. We do not want to modify anything in the predefined file; we simply want to copy that rule to another file and modify it there. For example, we chose to copy the **QuotedText** rule from the **normals.wml** file to a second file, named **custom-rules.wml**.
3. Edit the copy of the conversion rule to change its name. For example, we will give our new rule the name **QuotedTextItalic**. Use WML functions to modify the behavior of the rule.
4. As an initial test of whether you have written correct WML syntax, open the **custom-rules.wml** file in WebMaker. WebMaker will report any syntactic errors it finds.
5. Modify the WML file that contains mappings between paragraph tags and predefined conversion rules to include the file **custom-rules.wml**. In this example, the mappings file is named **manual.wml**. Make that WML file include the new file, **custom-rules.wml**. You can do that by using the WebMaker user interface, choosing the Includes view, and using **Edit>New Include**, or by using an ordinary text editor to edit **manual.wml**.
6. To test the conversion rule, make one of your paragraph tags map to the new rule, convert that document, and view the result in a browser.

10.1.1 Example: making quoted text appear in italics

In this example, we copy the predefined rule named **QuotedText** from the **normals.wml** file to a file named **custom-rules.wml**. The definition of the **QuotedText** rule is:

```
# Warning specifies that text be displayed as a BLOCKQUOTE,
# separated from text above and below by a blank line.
# Warning also displays the word "Warning" in bold before the
text.
PARAGRAPH "WarningBold" TYPE Normal
{
  CONTEXT Quote
  ACTIONS
  {
    write(*,concatenate(strong("Warning: "),
                        paragraph(text())));
  }
}
```

10.1.3 Example: Creating a level 4 bullet item rule

In this example, we want to have a rule for level 4 bullets. The closest predefined rule is **`L3BulletItem`**, which we find in the **`lists.wml`** file. The definition of the **`L3BulletItem`** rule is:

```
# L3BulletItem specifies level 3 bulleted list item.
PARAGRAPH "L3BulletItem" TYPE List
{
  LEVEL 3
  KIND Bullet
  ACTIONS
  {
    write(*,listitem(paragraph(text())));
  }
}
```

We copy the **`L3BulletItem`** rule and edit the copy. We change the name to **`L4BulletItem`**, edit the comment, and simply change the level from 3 to 4. The definition of the **`L4BulletItem`** rule is:

```
# L4BulletItem specifies level 4 bulleted list item.
PARAGRAPH "L4BulletItem" TYPE List
{
  LEVEL 4
  KIND Bullet
  ACTIONS
  {
    write(*,listitem(paragraph(text())));
  }
}
```

5. Modify the WML file that contains mappings between paragraph tags and predefined conversion rules to include the file **`custom-nodes.wml`**, and not to include the file you copied (in this example, **`nodesBTI.wml`**).
6. To test the node rules, convert the document and view the result in a browser.

10.2.1 Example of changing the header and footer

In this section, we copy the **`LevelOne`** node rule from **`nodesBTI.wml`** to a new file, **`custom-nodes.wml`**. The original definition of the **`LevelOne`** node rule appears in Figure 10.1. We modify the copy of that node rule to change the appearance of the header and the footer. In the header, we add our company's logo file followed by a horizontal rule above the navigation buttons. In the footer, we remove the phrase "Generated by Harlequin" and add our copyright notice, "Copyright (c) 1996 ACME, Inc. All rights reserved."

Note that we must place a copy of the graphic file containing the company's logo (**`acme-logo.gif`**) into the directory containing the HTML output, so that the HTML files that reference that graphic will find it.

```
NODE LevelOne
{
  TITLE concatenate(headingnumber()," ",headingtext())
  HEADER
  {
    @NavPanel=concatenate(
              button(
                     image("next.gif",BOTTOM),
                     filename(next),
                     image("nextg.gif",BOTTOM))," ",
              button(
                     image("prev.gif",BOTTOM),
                     filename(previous),
                     image("prevg.gif",BOTTOM))," ",
              button(
                     image("top.gif",BOTTOM),
                     filename(top),
                     image("top.gif",BOTTOM))," ",
              button(
                     image("content.gif",BOTTOM),
                     filename(TOC),
                     image("contentg.gif",BOTTOM))," ",
              button(
                     image("index.gif",BOTTOM),
                     filename(INDEX),
                     image("indexg.gif",BOTTOM))
              );
    write(*,image("acme-logo.gif",TOP));
    write(*,hrule());
    write(*,paragraph(@NavPanel));
    write(*,paragraph(maintitle()));
  }
  FOOTER
  {
    write(*,toc(1,local));
    write(*,hrule());
    write(*,address(concatenate(maintitle()," - ",date())));
    write(*,paragraph(@NavPanel));
    write(*,paragraph("Copyright (c) 1997 ACME, Inc.  All rights
          reserved."));
  }
}
```

Figure 10.2 Revised definition of LevelOne node rule (changes underlined).

nodesB.wml Contains node rules for a first page, level one, and level two and lower page templates. You should use this file if you use headings that create new nodes, and if you do not want to create a table of contents or index. This uses GIF files for images for the navigation buttons.

nodesBTI.wml Contains node rules for a first page, level one, level two and lower, table of contents, and index page templates. You should use this file if you use headings that create new nodes, and if you want to create a table of contents or index. This uses GIF files for images for the navigation buttons.

nodejava.wml In addition to the table of contents and index provided by **nodesBTI.wml**, this nodes file specifies a Java applet that creates a live table of contents on each HTML output page, which you can click on and use to go to any other page in your web. This uses GIF files for images for the regular navigation buttons.

normals.wml Contains the paragraph rules documented in Section 11.3, "Normals".

lists.wml Contains the paragraph rules documented in Section 11.4, "Lists".

headings.wml Contains the paragraph rules documented in Section 11.5.2, "Body headings".

contents.wml Contains the paragraph rules documented in Section 11.5.1, "Table of contents and related rules".

In addition to these files, you will find the files **heading2.wml** and **list2.wml**. See Section 11.6, "Additional files" for more information on these files.

11.2.3 The nodesTI.wml file

The **nodesTI.wml** file contains definitions of node templates that include those in **nodes.wml**, and adds templates for table of contents and index nodes. It uses text for navigation links.

11.2.4 The nodesBTI.wml file

The **nodesBTI.wml** file is identical to the **nodesTI.wml** file except that it uses images for navigation buttons rather than text. If you use **nodesBTI.wml** you must include the following image files in the same directory as the HTML files of the web:

```
content.gif      index.gif
contentg.gif     indexg.gif
next.gif         up.gif
nextg.gif        upg.gif
prev.gif         top.gif
prevg.gif        topg.gif
```

The "g" version of each image file is the greyed-out version for an unavailable action.

11.2.5 The nodejava.wml file

The **nodejava.wml** file is identical to the **nodesBTI.wml** file except that it specifies the creation of a Java applet which creates a live table of contents on each HTML output page, which you can click on and use to go to any other page in your web. If you use **nodejava.wml** you must include the following image files in the same directory as the HTML files of the web:

```
content.gif      index.gif
contentg.gif     indexg.gif
next.gif         up.gif
nextg.gif        upg.gif
prev.gif         top.gif
prevg.gif        topg.gif
```

In addition, you must also include the following Java class files in the same directory as the HTML files of the web:

NumberedBody *Normal*

Specifies a simple body paragraph preceded by a number and a space. The number is the character or characters created by FrameMaker autonumbering for this item. This rule is the same as **Default**.

PreformattedText *Normal*

Specifies that all line breaks, spaces, and tabs of the source text be preserved in the display. Display is in a fixed-width font like Courier.

QuotedText *Normal*

Specifies that text be displayed as a **BLOCKQUOTE**, separated from text above and below by a blank line.

SimpleBody *Normal*

Specifies a simple body paragraph.

11.4 Lists

Lists rules specify four kinds of list and hierarchy levels:

Glossary	Specifies a definition list item. Characters before a tab display as the definition term on the left. Characters on the right of a tab display as the definition text in a block on the right. Item numbers specify styles of glossary display with regular and bold text and numbers.
Bullet	Specifies an unordered list item preceded by a bullet character. Levels specify indentation and changing bullet character for nesting.
Number	Specifies a numbered list item proceeded by an Arabic number. WebMaker creates the numbering rather than use any in the FrameMaker source. Levels specify amount of indentation and new numbering stream for nested lists.

L1AutonumberItem *List*

Specifies level 1 list item with FrameMaker autonumber at the front.

L2AutonumberItem *List*

Specifies level 2 list item with FrameMaker autonumber at the front.

L3AutonumberItem *List*

Specifies level 3list item with FrameMaker autonumber at the front.

L1BulletItem *List*

Specifies level 1 bulleted list item.

L2BulletItem *List*

Specifies level 2 bulleted list item.

L3BulletItem *List*

Specifies level 3 bulleted list item.

L1NumberItem *List*

Specifies level 1 numbered list item.

L2NumberItem *List*

Specifies level 2 numbered list item.

L3NumberItem *List*

Specifies level 3 numbered list item.

L1PlainItem *List*

Specifies level 1 plain list item.

IndexHeading-IndexNode *Heading*

Level 1 in the hierarchy.

Heading 1 display size.

Index node, letter. Index entries are sorted alphabetically and displayed with headers dividing the entries into sections for each letter.

Identical to IndexHeadingL-IndexNodeL; provided for compatibility.

Only one index rule should be used for a document or book to produce an index node or nodes. A tag mapped to this rule, such as WWW-IX, should appear only once in the FrameMaker document or book.

Include **nodesTI.wml** or **nodesBTI.wml** from the library to create an index.

IndexHeadingL-IndexNodeL *Heading*

Level 1 in the hierarchy.

Heading 1 display size.

Index node, letter. Index entries are sorted alphabetically and displayed with headers dividing the entries into sections for each letter.

Only one index rule should be used for a document or book to produce an index node or nodes. A tag mapped to this rule, such as WWW-IX, should appear only once in the FrameMaker document or book.

Include **nodesTI.wml** or **nodesBTI.wml** from the library to create an index.

IndexHeadingN-IndexNodeN *Heading*

Level 1 in the hierarchy.

Heading 1 display size.

Index node, multiple nodes. Index entries are sorted alphabetically and displayed with a separate node for each letter section.

Table 11.1 Behavior of Body Heading Rules

Rule	Level	HTML	Horiz. Rule	New Node	Buttons
L1H1HR-NodeTop	1	<H1>	No	No	Next, Previous, Top
L1H1-NodeTop	1	<H1>	No	Yes	Next, Previous, Top
L2H1HR-NodeLower	2	<H1>	Yes	Yes	Next, Previous, Top, Up
L2H1-NodeLower	2	<H1>	No	Yes	Next, Previous, Top, Up
L2H2-NodeLower	2	<H2>	No	Yes	Next, Previous, Top, Up
L3H1HR-NodeLower	3	<H1>	Yes	Yes	Next, Previous, Top, Up
L3H1-NodeLower	3	<H1>	No	Yes	Next, Previous, Top, Up
L3H2-NodeLower	3	<H2>	No	Yes	Next, Previous, Top, Up
L3H3-NodeLower	3	<H3>	No	Yes	Next, Previous, Top, Up

Table 11.1 Behavior of Body Heading Rules

Rule	Level	HTML	Horiz. Rule	New Node	Buttons
L3H3	3	<H3>	No	No	
L3H3HR	3	<H3>	Yes	No	
L4H1	4	<H1>	No	No	
L4H1HR	4	<H1>	Yes	No	
L4H2	4	<H2>	No	No	
L4H2HR	4	<H2>	Yes	No	
L4H3	4	<H3>	No	No	
L4H3HR	4	<H3>	Yes	No	
L4H4	4	<H4>	No	No	
L4H4HR	4	<H4>	Yes	No	

11.5.1 Table of contents and related rules

The table of contents rules are defined in the **contents.wml** library file. This file also contains example rules for creating similar kinds of rules, for list of figures, list of tables, list of markers and so on; see Section 12.10, "List of figures or tables in WML"

When you want to create a mapped table of contents (see Section 3.6.2, "Mapped table of contents"), you map the paragraph rules of the FrameMaker table of contents to the table of contents rules. For example, if your table of contents contains entries for Chapter and 1Heading, the paragraph tags are ChapterTOC and 1HeadingTOC. You could map the ChapterTOC paragraph tag to the TOC-Entry-1 rule and the 1HeadingTOC paragraph rule to the TOC-Entry-2 rule. These TOC-Entry rules vary only in the level of indentation.

Each of these rules work for a table of contents that has titles in one column and page numbers at the far right margin. These rules strip the last token from

LOT-Entry-1 *List*

Specifies a list of tables entry with level 1 of indentation.

11.6 Additional files

You will find two other files in your library directory: **heading2.wml** and **list2.wml**. These files contain rules for documents that are more complex than typical documents, providing heading and list rules to levels deeper than those provided in **headings.wml** and **lists.wml**. Most WebMaker users will not need to use these files, but users with complex documents will appreciate the choices the additional levels provide.

11.6.1 The heading2.wml file

The file **heading2.wml** provides all of the heading rules in **headings.wml** (described in Section 11.5, "Headings") plus the following body heading rules for level 5 headings:

L5H1-NodeLower	**L5H2HR**
L5H1HR-NodeLower	**L5H3**
L5H1	**L5H3HR**
L5H1HR	**L5H4**
L5H2-NodeLower	**L5H4HR**
L5H2HR-NodeLower	**L5H5**
L5H2	**L5H5HR**

Also included in **heading2.wml** are the following body heading rules:

L*n*H1-NodeLower	**L*n*H3**
L*n*H1HR-NodeLower	**L*n*H3HR**
L*n*H1	**L*n*H4**
L*n*H1HR	**L*n*H4HR**
L*n*H2-NodeLower	**L*n*H5**
L*n*H2HR-NodeLower	**L*n*H5HR**
L*n*H2	**L*n*H6**
L*n*H2HR	**L*n*H6HR**

Where *n* is 6, 7, 8, 9, and 10.

If you use **heading2.wml**, you do not need to use **headings.wml**.

12.1 Basic information about WML programs

12.1.1 The structure of a WML program

A WML configuration file is divided into the following parts:

1. Include statements. See Section 12.2, “Include files”.
2. Definitions of user-defined variables. See Section 12.3, “User-defined variables”.
3. Node rules, which typically include navigation panels. See Section 12.5, “Node rules”.
4. Paragraph rules. See Section 12.6, “Primary paragraph rules”.
5. Character rules. See Section 12.8.1, “Character rule syntax”.
6. Format override rules. See Section 12.8.2, “Syntax of format override rules”.

12.1.2 Case sensitivity in WML

WML is not case sensitive. The only exception is where FrameMaker paragraph tags is used; they must use the same case in WML as they use in the FrameMaker document.

12.1.3 Order of elements in WML file

The order of elements in the WML file is significant in these cases (and no others):

- If an element is defined more than once, the last definition takes precedence over any previous definitions. For more information, see Section 9.5, “Rules defined more than once”.
- User-defined variables must be declared before being used.
- A node rule must be defined before a **`Heading`** paragraph rule may use it.

In the standard syntax using double quotes (**"*filename*"**), *filename* may be either a relative pathname or an absolute pathname. If it is a relative pathname, the included file must be relative to the directory of the WML file you are editing.

Note that the only way to specify a relative pathname is to edit the WML file in a text editor. When you specify an include file using WebMaker interactively, the result is always an absolute pathname.

In the following example of a relative pathname, the **nodesTI.wml** file must be in the same directory as the WML file you are editing. The syntax for a relative pathname of an include file that is in the same directory as the WML file you are editing is the same on all platforms:

```
INCLUDE "nodesTI.wml"
```

In the following examples of relative pathnames, the **lib** directory must be in the same directory as the WML file you are editing:

A relative pathname on Unix:

```
INCLUDE "lib/nodesTI.wml"
```

A relative pathname on Windows:

```
INCLUDE "lib\nodesTI.wml"
```

A relative pathname on the Macintosh:

```
INCLUDE "lib:nodesTI.wml"
```

The following are examples of absolute pathnames.

Absolute pathnames on Unix:

```
INCLUDE "/nova/webmaker/lib/nodesTI.wml"
INCLUDE "~skeene/lib/nodesTI.wml"
```

An absolute pathname on Windows:

```
INCLUDE "c:\webmaker\lib\nodesTI.wml"
```

An absolute pathname on the Macintosh:

```
INCLUDE "Banana:Applications:WebMakerTM 2.1:lib:nodesTI.wml"
```

Most WML files that use only the standard libraries should have the following at the beginning of the files:

1. **The default class.** Each HTML element has a default class. For elements in the main body of the web page, the default class name is the same as the name of the paragraph tag or character tag in the FrameMaker document.The default class of elements in the header of the web page is **HEADER**. The default class of elements in the footer of the web page is **FOOTER**. The default class of elements in the navigation panel is **NAV-PANEL**. For example, if you have a paragraph tag named Bullet and you map it to **L1BulletItem**, your HTML will appear like this:

```
<LI CLASS=Bullet>Text of first bullet.
```

2. **The CLASS attribute of conversion rules.** The WML definitions of node rules, paragraph rules (including uses paragraph rules), and character rules can contain a **CLASS** attribute to change the default class of HTML elements produced by those rules. The node rules produce the HTML elements in the header and footer. If the CLASS attribute is provided, it specifies a new default class that overrides the default described in **1.** For example, you can create a rule such as this:

```
PARAGRAPH "MyL1BulletItem" TYPE List
{
  LEVEL 1
  KIND Bullet
  CLASS MyBullet
  ACTIONS
  {
    write(*,listitem(text()));
  }
}
```

Now, if you map your Bullet paragraph tag to the **MyL1BulletItem** rule, the HTML will appear like this:

```
<LI CLASS=MyBullet>Text of first bullet.
```

An example of a uses paragraph rule that specifies a class is:

```
PARAGRAPH "Bullet" TYPE List
{CLASS MyBullet USES "L1BulletItem"}
```

kinds of node rules: one typical example is that the first page of the web is usually a different kind of node than the nodes triggered by heading paragraphs.

The general format of a node rule is:

```
NODE nodename
{
   specification of attributes
      HEADER   {
        specification of what appears at the top of the node }
      FOOTER   {
        specification of what appears at the bottom of the node  }
}
```

The elements of a node rule are:

NODE, **HEADER**, and **FOOTER** are WML keywords.

nodename is the user-assigned name of the node rule.

specification of attributes is the assignment of values to various node attributes, which may currently be the HTML **TITLE** or **CLASS**. See Section 12.5.1, “Specification of attributes of a node rule”.

specification of what appears at the top of the node can be any combination of calls to WML functions to perform the actions desired when the new HTML file is created. See Section 12.5.2, “The header and footer of a node rule”.

specification of what appears at the bottom of the node can be any combination of calls to WML functions to perform the actions desired when the new HTML file is closed. See Section 12.5.2, “The header and footer of a node rule”.

commands provide the **class** keyword argument, which overrides the **CLASS** attribute of the node rule). See Section 12.4, "WML support for CSS".

TITLE is written into the HTML file of a node on creation.

12.5.2 The header and footer of a node rule

The **HEADER** and **FOOTER** sections specify respectively what is to appear at the top and at the bottom of the node. This may include:

- A navigation panel.

  These are composed using predefined functions as building blocks. The functions of particular interest are:

  button(string1, url, string2, class:classname)
  filename(node)
  image(url, keyword-args)

- Table of contents; hypertext links to child nodes.

  Tables of contents are provided by the function **toc(**depth, scope). This is then written out into an HTML file with a function call such as:

  ```
  write(*,toc(4,global))
  ```

- Other information that improves the functionality of the generated Web.

  This may, for example, specify the heading of the upper section to appear at the beginning, or a signature for the document that is to appear at the end of each node. Any of the functions documented in Section 12.12, "Predefined functions" may be used to compose this information.

The **HEADER** and **FOOTER** are written into the HTML file of the node at creation and at closure of the node, respectively.

12.5.3 Nodes triggered by a heading paragraph

How can a node type be described such that it will contain:

The topmost heading of the FrameMaker document (that is, the title of the printed document) may not be the first paragraph of the document.

The FrameMaker document may not necessarily contain a topmost heading.

To guarantee the generation of the entry page, including a user-specified HTML **TITLE**, WebMaker requires the following:

- The node name of **FirstPage** is reserved for the node that describes the entry page. If no **NODE FirstPage** is defined, a default one is created by WebMaker.
- The topmost heading of a document (the title of the printed document) must not trigger a new node and must be declared of **TYPE Normal**, as shown in the following example.
- The HTML **TITLE** of the entry page is specified in the Make a Web dialog, and on the batch-mode command line with the **-t** switch:

  webmaker ... **-t** "*The Title of the first page*" ...

  This string may be accessed in the configuration by calling the predefined function:

  maintitle()

The following excerpt from a WML configuration file describes an entry page for a generated web and the conversion rules for the paragraph of the topmost heading.

"*FMparatag*"	is the tag name of the FrameMaker paragraph tag. Because FrameMaker paragraph tags are case-sensitive, this name must match the case of the paragraph tag.
type	may be either **`Heading`**, **`List`**, or **`Normal`**.
specification of attributes	Assignment of values to various attributes, depending on *type*.
statement of conversion action	Any combination of calls to WML functions to perform the desired conversion action.

12.6.1 Declaration of paragraph type

It is mandatory for WebMaker that a FrameMaker paragraph rule is assigned a **`TYPE`** for this to be translated. There is no **`TYPE`** default. A FrameMaker paragraph rule may have one of the following three types:

`Heading`	FrameMaker paragraphs tags that indicate a hierarchy in the document contents. Only paragraphs of the **`Heading`** type may trigger a new node or be included in tables of contents.
`List`	FrameMaker paragraphs that are to be considered as list items in a list environment.
`Normal`	FrameMaker paragraphs that should not be considered as either a **`Heading`** or a **`List`**.

12.6.2 Specification of attributes of a paragraph rule

The **`Class`** and **`FileNameKey`** attributes can be given in any paragraph rule, regardless of its **`TYPE`**.

Glossary, for a Definition List.

Normal

Context — **Preformatted**, to be mapped to the **PRE** HTML environment. **Quote**, to be mapped to the **BLOCKQUOTE** HTML environment. **Address**, to be mapped to the **ADDRESS** HTML environment. If **Context** is not present, the paragraph is not within any of the three HTML environments.

Inlist — If provided, the paragraph is to be mapped into a list item, **LI**, environment. If not provided, the paragraph is not to be mapped into a **LI** environment.

12.6.3 Conversion actions of a paragraph rule

Specify the actions, using combinations of predefined functions and user-defined variables, to be taken when encountering a paragraph of the declared format tag. If only default actions are desired, then no rules are specified other than the paragraph type declaration and, possibly, the setting of attribute values.

Below are three examples:

```
PARAGRAPH "1Heading" TYPE Heading
  {
    NEWNODE BasicSection
    LEVEL 1
    ACTIONS
      {
        write(*,heading(1,text()));
        write(*,hrule());
      }
  }
```

For **Normal** type paragraph rules, if **Context Preformatted** is specified, the default action is:

```
{  write(*,text());  }
```

12.7 Uses paragraph rules

WML enables you to declare that a FrameMaker paragraph tag is to be mapped in exactly the same ways as another previously declared paragraph rule. For example:

```
PARAGRAPH "Chapter" TYPE Heading { USES "L1H1NodeTop" }
```

If **USES** is used, the previously declared paragraph rule must be of the same type.

Uses paragraph rules can use the **CLASS** attribute, which has the same effect as it does in primary paragraph rules. See Section 12.6.2, “Specification of attributes of a paragraph rule” and Section 12.4, “WML support for CSS”. For example:

```
PARAGRAPH "Chapter" TYPE Heading {CLASS Head USES "L1H1NodeTop"}
```

12.8 Character rules

A character rule maps a FrameMaker character tag to an HTML character highlight.

Format override rules map characters that have no character tag, but do have formatting applied to them in FrameMaker to an HTML character highlight.

12.8.1 Character rule syntax

Every character tag to be translated must be declared and associated with one of the highlights supported in HTML or to None. Remember that it is the browsing software that determines how each HTML highlight is interpreted. For example, if you are reading this document with a World Wide Web browser, the browser that you are presently using displays these highlights as follows:

ent combinations of these parameters to match characters with no tags but with formatting applied to them and map them to HTML highlights.

The syntax is:

```
CHARACTER
{
  ANGLE   "name-of-angle"
  FAMILY  "name-of-font-family"
  VAR     "name-of-variation"
  WEIGHT  "name-of-weight"
    MAP  HTML-highlight-name>
}
```

where:

ANGLE, FAMILY, VAR, WEIGHT

are keywords. At least one of these must appear. The order is not important. Any one may not appear more than once.

name-of-angle, name-of-font-family, name-of-variation, name-of-weight

are case-sensitive strings that specify the value of the corresponding formatting parameter.

HTML-highlight-name

may be one of: **Big**, **Bold**, **Cite**, **Code**, **Definition**, **Emphasise**, **Italic**, **Keyboard**, **None**, **Sample**, **Small**, **Strong**, **Strikethrough**, **Subscript**, **Superscript**, **Teletype**, **Underline**, **Variable.**

You can map a character tag to **None** to give those characters no HTML highlight.

Characters can match more than one format override rule. For example, consider characters whose angle is Oblique and family is Courier. Those characters match both these rules:

```
CHARACTER { ANGLE  "Oblique" MAP Italic }
CHARACTER { ANGLE  "Oblique"
            FAMILY "Courier"
            MAP     Variable}
```

```
NODE ExternalTOC
{
  TITLE concatenate(maintitle()," - TOC")
  HEADER { ... }
  FOOTER { write(*,toc(4,global)) }
           ...  }
}

PARAGRAPH "TOCChapter" TYPE Heading
{  NEWNODE ExternalTOC
   FILENAMEKEY exttoc
   LEVEL 1
   ACTIONS { write(*,heading(1,text()));
             write(*,hrule());  }
}
```

The generated table of contents may then be pointed to by the use of the function **filename(**node) by supplying the specified **FILENAMEKEY** as parameter, such as:

```
...
write(*,button("[Contents]",filename(exttoc)));
...
```

12.9.2 Mapped table of contents

For basic information about a mapped table of contents, see Section 3.6.2, "Mapped table of contents". We define a *mapped table of contents* as one that was created in FrameMaker, and whose entries are mapped to WML rules.

The best way to understand how a mapped table of contents works in WML is to look at the rules in the library files that support them:

- The library file **contents.wml** contains paragraph rules that are intended to be used for the paragraph tags that appear in the FrameMaker table of contents. Notice, in particular, that each of these rules specifies TOC as the **FILENAMEKEY**.
- The library files **nodesTI.wml** and **nodesBTI.wml** define nodes that have a Contents button, which links to the table of contents. The definition of the Contents button uses the filename function with the argument TOC.

It is important to understand how WebMaker determines which HTML file is the distinguished table of contents, that is, the one that is the target of the Con-

```
NODE FirstPage
{
  TITLE maintitle()
  HEADER
  {
    @NavPanel=
      concatenate(
        button("[Next]", filename(next), "[Next]",
               class: "NavPanel"),
        " ",
        button("[Contents]",filename(TOC), "[Contents]",
                class: "NavPanel"),
        " ",
        button("[Figures]",filename(LOF), "[Figures]",
                class: "NavPanel"),
        " ",
        button("[Index]", filename(INDEX), "[Index]",
                class: "NavPanel"),
        " ");
    write(*,paragraph(@NavPanel));
  }
```

Lastly, edit the WML file you are using to include **LOF.wml** and **nodes-LOF.wml**, and to exclude **nodesTI.wml** (or **nodesBTI.wml**).

12.11 Index in WML

The technique for creating an index is much like the technique for creating a generated table of contents. The function that generates an index is:

```
index(markertype, indexoption)
```

where:

markertype is the type of source FrameMaker marker. The possible values are:

2 or **Index**
3 or **Comment**
4 or **Subject**
5 or **Author**
6 or **Glossary**
11, **12**, …, **24**

```
NODE IndexNode
{
  TITLE concatenate(maintitle()," - Index")
  HEADER { ... }
  FOOTER {
            write(*,index(index,letter));
            ...  }

PARAGRAPH "INDEXChapter" TYPE Heading
{  NEWNODE IndexNode
   FILENAMEKEY index
   LEVEL 1
   ACTIONS { write(*,heading(1,text()));
             write(*,hrule());  }
}
```

The index may then be pointed to by the use of the function **filename(**node) by supplying the specified **FILENAMEKEY** as parameter, as shown below:.

```
...
write(*,button("[Index]",filename(index)));
...
```

12.12 Predefined functions

The WebMaker Language provides a number of predefined functions to aid in the specification of the generated Webs. These are grouped as follows:

12.12.1 General I/O functions

closefile(*LUV***)**

Parameters *LUV* is a logical unit variable.

Closes the already open file identified by *LUV*.

For example, to close the file which, when opened by **openfile()**, had been associated to **@tocfile**:

```
closefile(@tocfile) --> ""
```

openfile(*LUV, filename, mode***)**

Parameters

For example:

```
concatenate("a","b","c") --> "abc"
```

date()

Returns a character string containing the current date.

For example:

```
date() --> "3 May 1994"
```

lowercase(*string***)**

Parameters *string* is a character string.

Returns *string* in all lowercase.

For example:

```
lowercase("AbCdEf 567") --> "abcdef 567"
```

number(*currprev***)**

Parameters *currprev* is **current** or **previous**, indicating the current or previous paragraph.

Returns a character string with the FrameMaker autonumber of the current or previous paragraph. **number()** defaults to the current paragraph.

For example:

```
number(current) --> "6.7.2"
```

The returned value of 6.7.2 means that the paragraph currently being translated by WebMaker has autonumber evaluating to 6.7.2.

text(*currprev, index1, index2***)**

Parameters

currprev **current** or **previous**, indicating the current or previous paragraph.

butlast(*string*)

Parameters *string* is a character string.

Returns a string that is like the argument *string* with the last word removed, where a word is any sequence of characters separated by whitespace.

For example:

```
butlast("vanilla ice cream") --> "vanilla ice"
butlast("page 7") --> "page"
butlast("See page 3-2") --> "See page"
```

butlast strips the last word without disrupting any HTML markup in the *string* argument. For example:

```
butlast("<B>twin cities</B>) --> "<B>twin</B>"
```

This function is intended for use in a table of contents where a space, rather than a tab, was used to separate the page number from the text at the beginning of the line. In that context, **butlast** strips the page number out of the table of contents.

For example, a typical use of **butlast** in context is:

```
PARAGRAPH "1TitleTOC" TYPE List
{
  LEVEL 1
  KIND Glossary
  ACTIONS
  {
     write(*,glossterm(bold(butlast(text()))));
  }
}
```

If you knew that the format of the table of contents included words such as "page 3" at the end of the line, you could use **butlast** nested twice to remove both "page" and "3" from the end of the line.

```
butlast(butlast("Introduction page 7")) --> "Introduction"
```

```
headingtext(next) --> "Overview of the Software"
```

index(*markertype, indexoption***)**

Parameters:

markertype The type of the source marker, which is one of the following values:

2 or **Index**
3 or **Comment**
4 or **Subject**
5 or **Author**
6 or **Glossary**
11, 12, …, 24

indexoption May be one of the following values: **simple**, **letter**, or **node**.

simple creates single node index with letter groups separated only by a space.

letter creates a single node index with letter groups separated by letter headings.

node creates one node per letter group, in which case a letter button is generated in the top index node for each letter group for which there is at least one index entry. Node rules may only be specified for the top index node which is also the only node that may be accessed using filename(node).

Returns the character string that contains the HTML markup for the generated index and, if *indexoption* is **node**, creates all the sub index nodes corresponding to each letter entry.

For example:

```
index(index,simple)
```

For example:

```
address("MR", class:AD) --> "<ADDRESS CLASS=AD>MR</ADDRESS>"
```

You can supply the **class** argument to specify the name of the CSS class of the HTML elements produced by the WML function. The **class** argument is optional; if you do not provide it, the HTML elements will have a default class name, as described in Section 12.4, "WML support for CSS".

12.12.5 HTML markup functions

Most of the functions listed below accept an optional argument named **class**. For information about the **class** argument and the default values for it, see Section 12.4, "WML support for CSS".

address(*string***, class:***classname***)**

Parameters *string* is a character string. *classname* is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns the HTML markup of *string* as **ADDRESS**.

For example:

```
address("MR") --> "<ADDRESS CLASS=classname>MR</ADDRESS>"
```

break(class:*classname***)**

Parameters *classname* is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns the HTML markup for line break.

For example:

```
break() --> "<BR CLASS=classname>"
```

element(*tagname***, *string*, class:***classname***)**

Parameters *tagname* is a string that names an HTML command in which to enclose the string that is the second argument. *string* is a character string. *classname* is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns the HTML markup of *string* within the HTML markup named *tagname*. The element function is useful for creating text in HTML markup that is not otherwise supported in WebMaker (which might include browser-specific extensions). You can use **element** to produce any markup that uses the simple syntax: **<***tagname***>***text***</***tagname***>**.

For example, you can use **element** to produce the **blink** command, which WebMaker does not have a special function to produce:

```
element("blink" "toto") --> "<blink class:classname>toto</blink>"
```

glossterm(*string*, class:***classname***)**

Parameters *string* is a character string. *classname* is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns HTML markup of *string* as a Definition Term (within a Definition List).

For example:

```
glossterm("PT"); --> "<DT CLASS=classname>PT"
```

See also **glossdescription(string, class:classname)**.

glossdescription(*string*, class:***classname***)**

Parameters *string* is a character string. *classname* is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns HTML markup of *string* as a Definition Description (within a Definition List).

height:*integer*

alt:*string* (where *string* is the alternate text for the image)

border:*integer*

hspace:*integer*

vspace:*integer*

usemap:*string* (where *string* is a URL for an image map)

ismap:**ismap**

class:*classname* (where classname is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns HTML markup for the image file specified by *url* with the provided keyword arguments. For example:

```
image("next.gif", align:bottom, alt:"Next") -->
"<IMG ALIGN="BOTTOM" ALT="Next" SRC="next.gif" CLASS=classname>"
```

The syntax of image changed in WebMaker 3.0. The old syntax is still supported for compatibility with previous releases. The old syntax is:

image(*url, alignment*, **class**:*classname***)**

The *alignment* argument is **top**, **bottom**, **middle**, **left**, or **right**. For example:

```
image("example.gif",bottom) -->
"<IMG ALIGN="BOTTOM" SRC="example.gif" CLASS=classname>"
```

link(*string,url,* **class**:*classname***)**

Parameters *string* is a character string. *url* is a URL. *classname* is a string that names the CSS class of the HTML elements produced by this function. The **class** argument is optional.

Returns the HTML markup for the hypertext link with *string* as anchor text pointing to the address specified by *url*.

For example:

12.12.6 HTML character markup functions

The following functions accept as input a single character string and output the HTML markup for the string as highlighted text. These functions also accept an optional argument named **class**. For information about the **class** argument and the default values for it, see Section 12.4, "WML support for CSS".

big(*string*, **class**:*classname***)**

For example:

```
big("toto") -> "<BIG CLASS=classname>toto</BIG>"
```

bold(*string*, **class**:*classname***)**

```
bold("toto") -> "<B CLASS=classname>toto</B>"
```

cite(*string*, **class**:*classname***)**

For example:

```
cite("toto") -> "<CITE CLASS=classname>toto</CITE>"
```

code(*string*, **class**:*classname***)**

For example:

```
code("toto") -> "<CODE CLASS=classname>toto</CODE>"
```

definition(*string*, **class**:*classname***)**

For example:

```
definition("toto") -> "<DFN CLASS=classname>toto</DFN>"
```

emphasise(*string*, **class**:*classname***)**

For example:

```
emphasise("toto") -> "<EM CLASS=classname>toto</EM>"
```

superscript(*string***, class:***classname***)**

For example:

superscript("toto") -> "^{toto}"

teletype(*string***, class:***classname***)**

For example:

teletype("toto") -> "<TT CLASS=*classname***>toto</TT>"**

underline(*string***, class:***classname***)**

For example:

underline("toto") -> "<U CLASS=*classname***>toto</U>"**

variable(*string***, class:***classname***)**

For example:

variable("toto") -> "<VAR CLASS=*classname***>toto</VAR>"**

12.12.7 Java applet functions

applet(*keyword-args***)**

Parameters The *keyword-args* can be given in any order. They include:

applet-code:"*filename***"** where filename contains the code of the Java applet.

width:*integer* where *integer* is the width in pixels that the applet should consume.

height:*integer* where *integer* is the height in pixels that the applet should consume.

default:*string* where *string* is the text the browser should display if it does not support Java applets.

simple example 13
external links, including 41
ExtTOCHeading-TocNode rule 103

F

File > Close command 48
File > New command 48
File > Open command 48
File > Quit command 48
File > Save As command 48
File > Save command 48
File > Use RapidRules command 48
`filename` WML function 143
first page of web
WML syntax 123
FixedWidthText rule 99
FMDocumentTitle rule 99
footers and headers
changing 85, 89
footnotes 53
format override rules
WML syntax 131
format overrides 66
FrameMaker
documents 1
compared to WWW documents 2
mapping document elements to WML rules 28
markers of Type 25 41
style recommendations 65

G

generated table of contents 38
WML syntax 133
`glossdescription` WML function 148
`glossterm` WML function 148
graphics 36, 66
consider resolution 70
converted 36
converting 36
importing 70
in anchored frames 66, 67
navigation buttons 36
using conditional text for 70

H

headers and footers
changing 85, 89
headers and footers of web pages
generating 122
WML syntax 122
`heading` WML function 149
`heading2.wml` 96, 110
`headingnumber` WML function 143
headings rules 103, 105
heading 105
HR 105
level 105
node 105
use of node rules in 97
using in single-node webs 28
`headings.wml` 96
`headingtext` WML function 143
Help > About WebMaker command 49
Help > Help command 49
hierarchy 28, 69, 103
highlights of WebMaker 11
hotlinks, to external URLs 41
`hrule` WML function 149
HTML 2
hypertext markers 10

I

Ignore rule 99
`image` WML function 149
importing graphics 70
include file 78
include files
adding to WML files 35
choosing which library files to include 34
how to use 33
organizing 79
overview 78
relative and absolute pathnames in 79
WML syntax 115
index 40
example 9
generating in web documents 40
node rule 98
WML syntax 136
Index button 5, 9
`index` WML function 144
IndexHeading-IndexNode rule 104
IndexHeadingL-IndexNodeL rule 104
IndexHeadingN-IndexNodeN rule 104
IndexHeadingS-IndexNodeS rule 105
InListNumbered rule 99
InListSimple rule 99
Internet Explorer 3.0 bugs with CSS 57
`italic` WML function 153

nodejava.wml 96
nodejavawml 98
nodes 28
nodes.wml 95, 97
nodesB.wml 96, 97
nodesBTI.wml 96, 98
nodesTI.wml 95, 98
normals rules 99
normals.wml 96
number WML function 140
NumberedBody rule 100

O

Open WML button 47
openfile WML function 138
order of WML elements 82
output of WebMaker 4

P

paragraph rules
  overview 80
  primary and uses 80
  recommended mappings for FrameMaker tags 29
  technique for using 81
  WML syntax 125
paragraph tags
  mapping to paragraph rules 29
paragraph WML function 151
PreformattedText rule 100
Previous button 5
primary paragraph rules 80
  WML syntax 125
primary rule 80

Q

quick start 13
QuotedText rule 100

R

RapidRules **25**
  dialog 49
  table of contents behavior 39
RapidRules button 47
referencing external files in web documents 41
rules
  headings 103, 105
  lists 100
  nodes 97
  normals 99
rules defined more than once 82

S

sample output of WebMaker 4
sample WML function 153
Save WML button 47
saving a WML file 23
simple example 13
SimpleBody rule 100
single-node web 28
  WML file 28
small WML function 153
span WML function 151
special characters 41
starting WebMaker 15
Stop button 48
strikethrough WML function 153
strong WML function 153
style recommendations 65
  attach anchored frames to their own paragraphs 67
  check master and reference pages 69
  check order of text flows 69
  consider imported graphics resolution 70
  put graphics in anchored frames 66
  use autonumbering 68
  use cross references 67
  use heading levels consistently 69
  use paragraph and character catalogues 66
Style Sheet> option 53
subscript WML function 153
superscript WML function 154

T

table of contents 36
  example 8
  generated 38
  local 37
  mapped 37
  node rule 98
  RapidRules behavior 39
  rules for
    list of figures 108
  WML syntax 133
tables 36
teletype WML function 154
text WML function 140
time WML function 141
toc WML function 145

appletparameter 155
big 152
bold 152
break 146
butlast 142
button 147
center 147
cite 152
closefile 138
code 152
comment 147
concatenate 139
date 140
definition 152
element 148
emphasise 152
filename 143
glossdescription 148
glossterm 148
heading 149
headingnumber 143
headingtext 143
hrule 149
image 149
index 144
italic 153
keyboard 153
link 150
listitem 151
lowercase 140
maintitle 145
number 140
paragraph 151
sample 153
small 153
span 151
strikethrough 153
strong 153
subscript 153
superscript 154
teletype 154
text 140
time 141
toc 145
underline 154
variable 154
write function 139
write WML function 139
writing a file to MIF 14
WWW documents
comparing to FrameMaker documents 3
defined 2
including references to 41