WebMaker™ User Guide

Version 3.0

The combination of WebMaker and FrameMaker enables you to publish a document both as a printed document and as a World Wide Web document, from a single FrameMaker source. WebMaker converts FrameMaker documents and books to a hypertext network of HTML files, or a web, which may be viewed by World Wide Web browsers such as Netscape Navigator, Mosaic, or Internet Explorer.

We have provided two ways to print the *WebMaker User Guide*. You can print the Adobe Acrobat PDF file using the Adobe Acrobat Reader, which you can obtain at no cost from Adobe. Or, you can print the PostScript® file `wmug.ps` in the `doc` subdirectory of the WebMaker directory.

Copyright and Trademarks

WebMaker User Guide

Version 3.0

April 1997

Part number: WM30UG

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.

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/

Contents

1

Introduction

WebMaker converts FrameMaker documents into World Wide Web documents. With the combination of WebMaker and FrameMaker, you can publish in both printed and hypertext formats from a single source. This provides several advantages:

- Information is consistent in both formats.
- Maintenance is easier when all of your writing is in a single source.
- Authoring is easier in the FrameMaker WYSIWYG environment than in most WWW writing programs.
- Both the printed and WWW publications can look like they were created for the format in which they appear. The printed document has all of the typographic features available in a good desktop publishing package. The WWW document has the simpler formatting and hyperlinks that can be used in a World-Wide Web browser.

1.1 FrameMaker documents

FrameMaker is a desktop publishing package that creates documents that include text, graphics, tables, and equations. It allows organization in single files or in multiple-file books. Its automatic pagination updates index, table of contents, and cross references. It runs on PCs with Windows, Macintoshes,

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.

The **ADDRESS** and **BLOCKQUOTE** environments.

- Lists made up of list items, divided into two cases:

  The ordered and unordered lists, **OL** and **UL**. These contain the HTML list item, **LI**, which may contain simple paragraphs and other lists.

  The glossary list, **DL**. This is made up of pairs of **DT** (definition term) and **DD** (definition description). **DT** may contain everything that the basic paragraph **P** may contain. **DD** may contain everything that **LI** may contain.

- Text in table format. WebMaker converts your FrameMaker tables into simple HTML tables.

It is important to note that HTML does not offer the same page layout capability as FrameMaker. As a result, it is not always possible to reproduce the exact presentation of the original FrameMaker document with HTML.

1.3 WebMaker rules

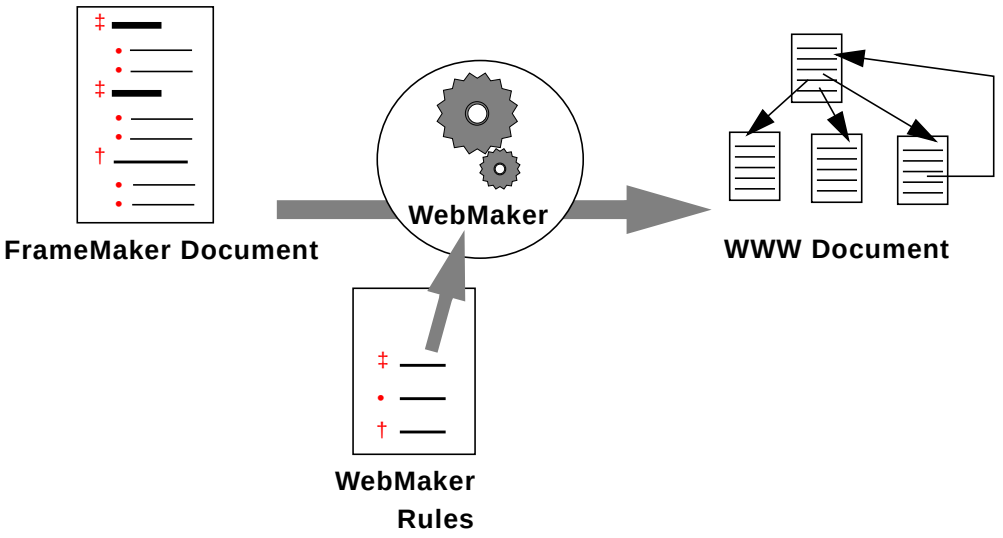

Figure 1.1 WebMaker rules control the conversion

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 1.2 Partial outline of *WebMaker User Guide*

You can choose any of the following ways to present this material:

- All on one HTML page
- Each chapter on its own HTML page.
- Each chapter and section on its own HTML page.
- Each chapter, section, and subsection on its own HTML page.

We decided on the final choice: to make each chapter, section, and subsection appear on its own HTML page. The next question to answer is — how can readers navigate from one part of the document to another? We provided several ways to navigate the document: navigation buttons, a detailed table of contents (containing links the reader can click on, to go directly to any chapter, section, or subsection), an index (containing links the reader can click on, to follow the index entry to the section where it occurs in the document), and cross references (which the reader can click on, to go to the section that is referenced).

1.4.2 Navigation buttons in the web document

Figure 1.3 shows the first page of the *WebMaker User Guide*, as it appears in a browser.

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

have navigation buttons at the top of the web page, and having them at the bottom of the web page is useful for pages that are larger than one screen.

The first page of the *WebMaker User Guide* contains some text, and after the text it contains links to the top-level sections of the web document. You can click on any of these links to go to that section. Clicking on the link labeled "Contents" has the same effect as clicking on the **Contents** navigational button. Similarly, clicking on the link labeled "Index" has the same effect as clicking on the **Index** navigational button.

Figure 1.4 shows another page in the *WebMaker User Guide*.

Figure 1.4 Section 1.4 of the *WebMaker User Guide*

The page in Figure 1.4 has additional navigational buttons: **Previous**, **Up**, and **Top**. The **Previous** button brings you to the page that occurs immediately

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 1.5 Contents of the *WebMaker User Guide*

WebMaker supports different kinds of tables of contents. The one shown in Figure 1.5 is a generated table of contents, which WebMaker generated automatically, based on the headings in your document. Another kind is a mapped table of contents, which reflects exactly the FrameMaker table of contents, but strips away the page numbers and converts the entries to hypertext links. For more information, see Section 3.6, “Table of contents”.

1.4.4 Index

Figure 1.6 shows the a portion of the index of the *WebMaker User Guide*, which is the page you see if you click on the **Index** 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.

Similarly, WebMaker converts FrameMaker's hypertext markers to HTML links. The following types of FrameMaker's hypertext markers are converted: **gotolink**, **gotolinkfitwin**, **openlink**, **openlinkfitwin**, **gotoObjectId**, and **openObjectId**. Each of these hypertext markers must have a corresponding **Newlink** marker that is the target of the link.

1.5 Highlights of WebMaker

WebMaker offers the following features:

- Ability to publish a FrameMaker document simultaneously in hardcopy and on the WWW with a single set of sources.
- Easy introduction to conversion, by using the Wizard.
- Support for Cascading Style Sheets.
- Superior navigational features. The FrameMaker document can be converted into a single HTML page, or many HTML pages. The rule that you choose for each FrameMaker paragraph tag controls whether or not that kind of paragraph starts on a new HTML page. The hierarchy of your FrameMaker document is preserved in the web document.
- Automatic generation of conversion rules for your documents using libraries of predefined rules, to get you started quickly. This feature is called RapidRules.
- Automatic generation of a detailed table of contents, if desired.
- Automatic creation of a mapped table of contents, list of tables, or list of figures, that corresponds to the FrameMaker table of contents, list of tables, or list of figures.
- Automatic creation of an index, if desired. The index is created from the Index tags in the FrameMaker document.
- Automatic insertion of navigational buttons.
- Automatic conversion of cross references and hypertext markers to HTML links.
- Automatic handling of graphics, tables, and equations.

- 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).

2

Quick Start

In this section, we show how to get started quickly with WebMaker, by taking a small FrameMaker document and converting it to a web document. Once you have converted the sample document, you can follow the same steps with your documents. The basic steps we follow in this chapter are:

1. Write the FrameMaker document to MIF format.
2. Start WebMaker.
3. Use RapidRules to create a WML file for the document.
4. Make the web document.
5. Browse the web document.
6. Save the WML file.

2.1 Sample document

In this chapter, we convert a simple FrameMaker document, **`simple.doc`**. It is available in the **`examples`** subdirectory of the **`webmaker`** directory. This document contains eight FrameMaker paragraph tags and two character tags. It is one page long. Figure 2.1 shows the document.

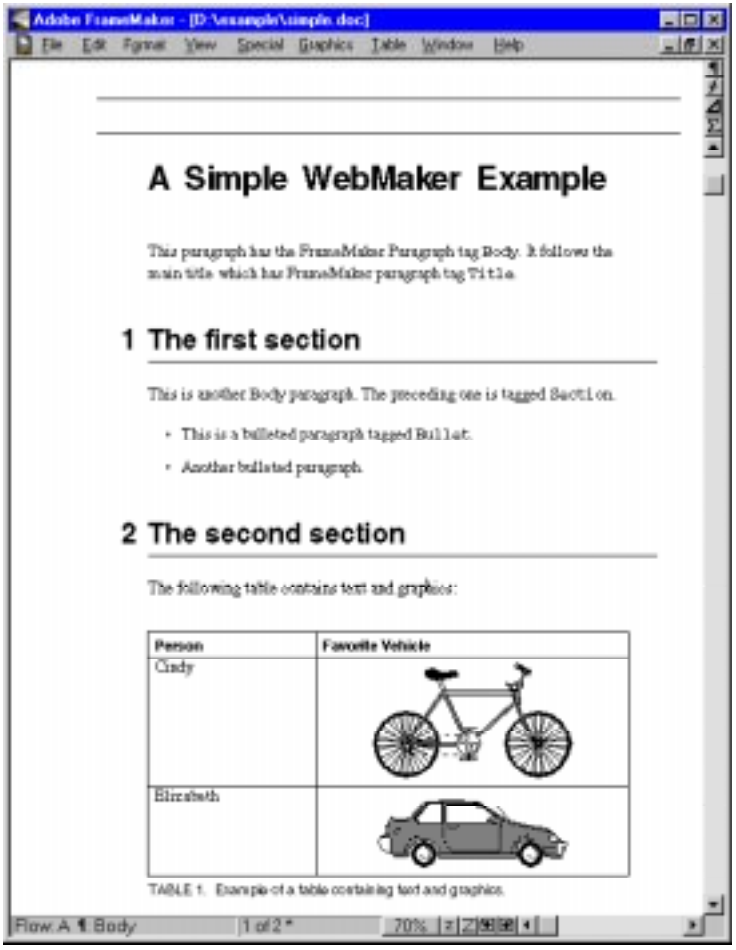

Figure 2.1 FrameMaker file: **`simple.doc`**

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:

- Open the file in FrameMaker. The filename of our sample document is **`simple.doc`**.
- In FrameMaker, choose **File > Save As**.
- In the FrameMaker Save File dialog, specify the MIF format, and the filename, providing the extension **`.MIF`**.

If you are converting a book to HTML, you need to save each file in the book, and the book file itself, to MIF. The filename of the MIF of the book file can be anything ending in **`.MIF`**, such as ***`name`*`.MIF`**.

If you are using UNIX, Appendix B of your *Using FrameMaker* book provides information on the UNIX program **`fmbatch`**. The **`fmbatch`** program is provided with FrameMaker; it can make the process of saving book files as MIF somewhat easier.

2.3 Start WebMaker

To start WebMaker on UNIX:

- Give the following UNIX command: **`webmaker`**

To start WebMaker on Windows 95:

- Click **Start**.
- Choose **Programs**.
- Choose **Harlequin WebMaker**.
- Choose **WebMaker 3.0**.

To start WebMaker on the Macintosh:

- Open the WebMaker folder by double-clicking on it.
- Double-click on the WebMaker icon.

The WebMaker Wizard will appear, as shown in Figure 2.2.

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. Click **Browse** and specify the MIF file to convert; it can be a MIF file of a single document file or a book. Look in the WebMaker directory for the **`examples`** directory, and specify the file in that directory named **`simple.MIF`**.
3. Click **Multiple Web Pages** and then click **Do It**.
4. Click **View WML Rules** and then click **Do It**. The main window appears, showing the results of running RapidRules on the **`simple.MIF`** document. See Figure 2.3.

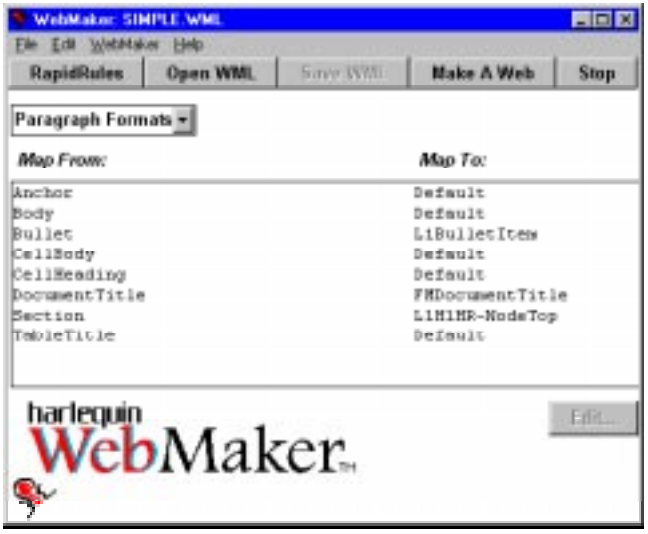

Figure 2.3 WebMaker main window

Each paragraph tag appears in the left column, and the rule it uses appears in the right column. You can explore other aspects of the WML file by selecting another view. Click on the drop-down list where **Paragraph Formats** appears to see the other available views: **Character Formats**, **Format Overrides**, and **WML Libraries**.

For more information about RapidRules, see Section 4.3, “RapidRules dialog”.

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.

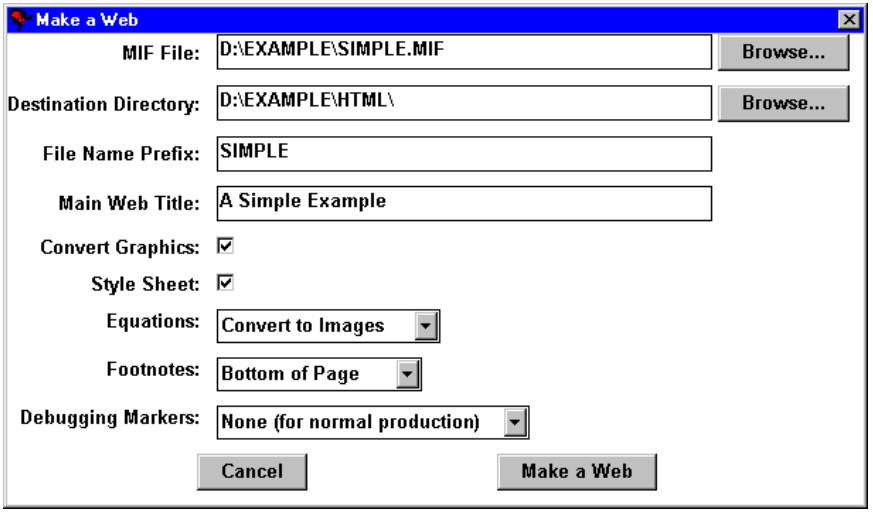

Figure 2.4 Make a Web dialog

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”.

2.6 Browse the web document

- Start a World Wide Web browser such as Netscape Navigator or Communicator, Microsoft's Internet Explorer, or NCSA Mosaic. Note that WebMaker creates a style sheet and uses it in the HTML files, so you will see differences in the output when you use a browser that supports CSS and one that does not. See Chapter 5, "Cascading Style Sheets".
- Give the command to open a local file (such as **File > Open Local** in Mosaic, or **File > Open File** in Netscape Navigator), and provide the filename of the web document.

  This filename is a combination of the destination directory you specified, the prefix (if any), and the filename of the first file. In this case, RapidRules has defaulted **`html`** as the destination directory, and suggested **`simple`** as the filename prefix; therefore, the filename of the first page of the web document would be:

  `/webmaker/html/simple-1.html` on UNIX

  `C:\webmaker\html\simple_1.htm` on Windows

  `Banana:WebMaker:html:simple-1.html` on the Macintosh, assuming the hard drive is named Banana

The following three figures show how the web document appears when viewed in a browser that does not support CSS. WebMaker converted the file **`simple.MIF`** into three web pages. Figure 2.5 shows the first page of the web document.

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 2.6 Second web page of A Simple Example

The second page has three navigation buttons (which appear at the top and at the bottom of the page). **Next** goes to the next page; **Previous** goes to the previous page; and **Top** goes to the first page of this web document.

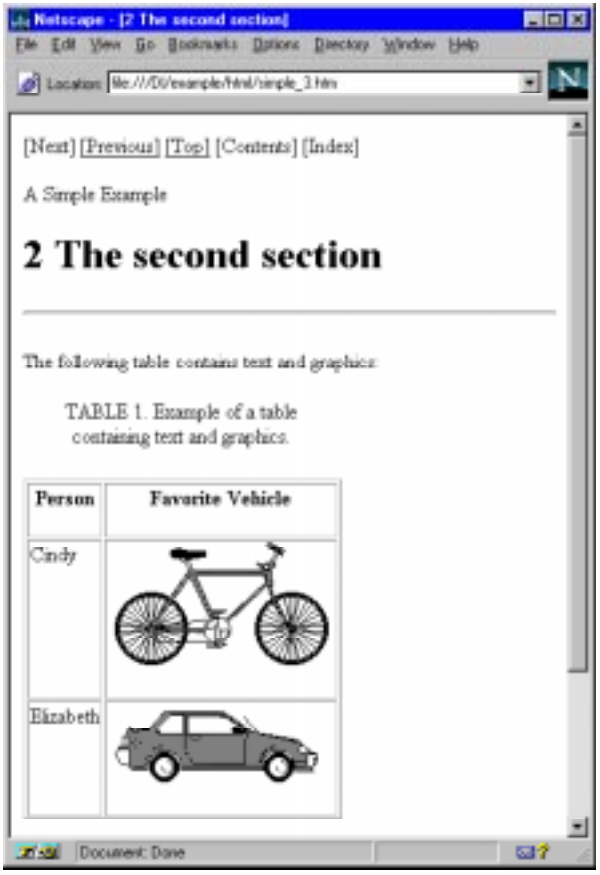

Figure 2.7 Third web page of A Simple Example

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.

2.7 Save the WML file

It is important to save the WML file. A WML file is intended to be used many times. If you intend to experiment further with the **`simple.MIF`** document, then you should save this WML file as **`simple.wml`**. In our example, RapidRules provides this name as the default name for the WML file.

To save, in the main window, click **Save WML**.

When you begin to convert real documents, typically you will be continuing to edit the FrameMaker document, and you will want to make the web again when you have a new version of the document. Also, you might have many documents that use the same FrameMaker template, and you might want to make a web for any number of those documents. You can use the saved WML file to convert the same document multiple times, or to convert multiple documents that use the same FrameMaker template.

2.8 Quick start with your document

We recommend that you convert the document immediately, view the results, and then make changes to improve the results. The steps of the initial conversion are covered in this chapter. We summarize them here:

1. Write the FrameMaker file to MIF format. See Section 2.2, “Write the FrameMaker file to MIF format”.
2. Start WebMaker. See Section 2.3, “Start WebMaker”.
3. Use RapidRules and name the MIF file you wish to process. Section 2.4, “Use RapidRules”.
4. Click on **Make a Web** and generate the web document. For information about each of the options in this dialog, see Section 4.4, “Complete options for making a web”.
5. Save the WML file. See Section 2.7, “Save the WML file”.
6. Browse the web document. See Section 2.6, “Browse the web document”.
7. Improve the results. See Chapter 3, “Improving the Results”.

3

Improving the Results

RapidRules is an automatic one-button tool to help you get started quickly on document conversion. It makes educated guesses at the proper WML rule mappings for a given FrameMaker document or book and produces a WML file with those mappings for you. RapidRules cannot always guess perfectly. At this point, you can start to improve the results in a number of ways:

1. The first step is to identify which paragraphs do not look right in the HTML output. You can then map those paragraphs to different paragraph rules. You can also change the mappings of character tags to character rules. See Section 3.1, “Changing mappings” and Section 3.2, “Mapping recommendations”.
2. If your document contains characters that have formats applied to them, but do not have character tags, you can provide an format override rule that specifies how the characters should appear. See Section 3.3, “How to use format override rules”.
3. Include the appropriate libraries in the WML file. See Section 3.4, “How to use include files”.

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”.

3.1 Changing mappings

3.1.1 Changing paragraph mappings

To change the mapping of a paragraph tag in WebMaker:

1. Choose the **Paragraph Formats** view in the main window.
2. Select the paragraph tag.
3. Click **Edit**.
4. Choose the appropriate category of rules: Heading, List, or Normal.
5. If necessary, choose the subcategory. For example, the List category has three subcategories: Numbered, Bullet, and Glossary.
6. Choose the rule itself.
7. Click **OK**.
8. When you are finished, remember to save the WML file by clicking **Save WML**.

3.1.2 Changing character mappings

To change the mapping of a character tag in WebMaker:

1. Choose the **Character Formats** view in the main window.
2. Select the character tag.
3. Click **Edit**.
4. Choose the appropriate character rule.
5. Click **OK**.
6. When you are finished, remember to save the WML file by clicking **Save WML**.

3.2 Mapping recommendations

In this section, we provide some advice for the overall structure of your WebMaker web and list recommendations for mapping the most commonly

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.

3.2.2 Paragraph format mappings

The table below lists various paragraph types you might have in a FrameMaker document and the WML rules we recommend for both multiple– and single–node webs. All the WML rules listed can be found in the library files distributed with WebMaker.

Table 3.1 Paragraph format mappings

Paragraph type	WML rule	
	Multiple Nodes	Single Nodes
body text	Default	same
bulleted list	L1BulletItem	same
bulleted list text (non-bulleted text indented at same level as bullet list)	L1PlainItem	same
bulleted list: second level	L2BulletItem	same
bulleted list text: second level (non-bulleted text indented at same level as bullet list)	L2PlainItem	same
code example	PreformattedText	same
definition list	L1GlossaryItem-1	same
document title	FMDocumentTitle	same
document subtitle	Default or L2H2	same
figure, table captions	Default	same
footers of a page	Ignore	same
glossary item	L1GlossaryItem-2	same

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

Table 3.1 Paragraph format mappings

Paragraph type	WML rule	
	Multiple Nodes	Single Nodes
table of contents formats (such as HeadingTOC)	Ignore (for a generated TOC) or TOC-Entry-1, TOC-Entry-2 (for a mapped TOC)	same
table of contents title	ExtTOCHeading-TOCNode (for a generated TOC) or L1H1-NodeTop (for a mapped TOC)	same

3.2.3 Character format mappings

The table below lists various character formats you might have in a FrameMaker document along with our recommended HTML highlights for each.

Table 3.2 Character format mappings

Character type	HTML highlight
bold	Strong, Bold
code example	Code, Keyboard, Teletype
definition term	Definition
italic	Emphasise, Italic
reference	Cite
sample or quoted text	Sample
underlined	Underline
variable name	Variable or Italic

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”).

3.3.1 Adding a new format override rule

To add a new format override rule in WebMaker:

1. Choose the **Format Overrides** view in the main window
2. Choose **Edit > Add**.
3. Enter the characteristics of the format override rule, including Angle, Weight, Variation, and the Character Rule to use for characters that match this rule.
4. Click **OK**.
5. When you are finished, remember to save the WML file by clicking **Save WML**.

3.3.2 Changing the mapping of a format override rule

To change the mapping of a format override rule in WebMaker:

1. Choose the **Format Overrides** view in the main window.
2. Select the format override rule to edit.
3. Click **Edit**.
4. Choose the Character Rule to use.
5. Click **OK**.
6. When you are finished, remember to save the WML file by clicking **Save WML**.

3.4 How to use include files

WebMaker provides a library of WML files that can be used to convert most documents. The library files are stored in the **`lib`** subdirectory of your WebMaker directory. This section describe techniques for using the library files in conjunction with WML files you create.

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”.

nodesBTI.wml 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 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”.

nodejava.wml You should use this file if you want the functionality of **nodesBTI.wml** along with a small 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 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”. You will also need a set of Java class files, copied from the **lib** directory to the directory where the HTML files are stored. See Section 11.2.5, “The nodejava.wml file”.

If your document will contain a mapped table of contents, include this file:

contents.wml You should use this file if you want to create a mapped table of contents. See Section 3.6.2, “Mapped table of contents”.

3.4.2 Adding an include file to a WML file

To add an include file in WebMaker:

1. Choose the **WML Libraries** view in the main window.
2. Choose **Edit > Add**.
3. Use the pathname dialog to select the include file.
4. Click **OK**.
5. When you are finished, remember to save the WML file by clicking **Save WML**.

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.

3.6.1 Local table of contents

A *local table of contents* is a list of links to subtopics within in a topic. For example, a node that begins with a level-2 header displays the level-3 header subtopics. When you include one of the **`nodes.wml`**, **`nodesB.wml`**, **`nodesTI.wml`**, or **`nodesBTI.wml`** library files in your WML file, each node of the web document includes a local table of contents at the bottom of the page.

To have WebMaker create a local table of contents:

- Include any one of the node rules files from the library in your WML file.
- Map some of your paragraph tags to rules of type Heading.

3.6.2 Mapped table of contents

A *mapped table of contents* is a node that contains a table of contents derived from the FrameMaker table of contents; it is a table of contents for the whole web of the FrameMaker document. Each entry in the mapped table of contents exists in the FrameMaker table of contents; the paragraph tags are mapped to TOC-Entry rules.

The **`contents.wml`** library file contains TOC-Entry rules that create a mapped table of contents.

To have WebMaker create a mapped table of contents:

- Your FrameMaker document must have a table of contents. When you generate that table of contents in FrameMaker, you must be sure that the Create Hypertext Links box is checked.
- Typically, you want the mapped table of contents to start on a new web page. To make this happen, your FrameMaker document must have a paragraph that acts as the title of the table of contents, in the same flow as the body of the table of contents, and you must map that paragraph tag to a rule that starts a new web page, such as L1H1-NodeTop.
- Include the **`contents.wml`** library file in your WML file. (If you use RapidRules on a book file that includes a FrameMaker table of contents, it will include this library file for you.)

- 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

of contents may differ from the FrameMaker table of contents since the tags used in FrameMaker may be different.

To have WebMaker create a generated table of contents:

- Include a library file such as **`nodesTI.wml`** and **`nodesBTI.wml`** in your WML file. These library files define node rules that create a generated table of contents and index.
- Include the **`headings.wml`** library file in your WML file. This file defines a paragraph rule that uses a generated table of contents node rule.
- Create a paragraph tagged with a special tag, such as WWW-TOC, exactly once in the FrameMaker document. This paragraph can be the table of contents title, but it must be a unique tag rather than something used elsewhere in the document, such as Chapter or 1Heading.
- Map the table of contents tag to the **`ExtTOCHeading-TocNode`** rule.
- Map the paragraph tags from the FrameMaker table of contents (if any) to the Ignore rule. For example, map tags named ChapterTOC and 1HeadingTOC to the Ignore rule.

You can write customized WML to create tables of contents that use a depth of other than 1 level for local table of contents and 4 levels for a generated table of contents. See the WML node function **`toc(depth, scope)`**.

The navigation panels provided in the **`nodesTI.wml`** and **`nodesBTI.wml`** library files include a link that takes a user to the generated or mapped table of contents from other nodes.

Note: If you include **`nodesTI.wml`** or **`nodesBTI.wml`** in your WML file and you do not map a paragraph tag from your FrameMaker document to one of the TOC or Index WML rules, you could end up with HTML files that have greyed-out Contents and Index buttons. You can remove the Contents and Index buttons by using the **`nodes.wml`** or **`nodesB.wml`** files instead. See Section 3.10, “Troubleshooting RapidRules output” for more information.

3.6.4 How RapidRules handles table of contents

If RapidRules finds a paragraph tag named WWW-TOC, it creates a generated table of contents.

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.

Multiple nodes index

Creates multiple nodes. Index entries are sorted alphabetically and displayed with a separate node for each letter section. The **IndexHeadingN-IndexNodeN** rule in **headings.wml** produces this.

The rules in the **nodeTI.wml** and **nodesBTI.wml** library files create the index entries from the FrameMaker Index markers. You can write WML index rules that use any of the FrameMaker markers (except markers of Type 25, which are used specially by WebMaker; see Section 3.8, "Including external links"). Index rules use the WML node function **index(markertype, indexoption)**.

The navigation panels provided in the **nodesTI.wml** and **nodesBTI.wml** library files include links that take a user to the index from other nodes.

3.8 Including external links

You can include links to any local file or WWW URL in the FrameMaker document. Use one marker of Type 25 on each side of the text that represents the link. The text of the first marker contains the filename or URL. Here are three examples:

```
EXTERNAL view-me.html

EXTERNAL ~/public_html/view-me.html

EXTERNAL http://www.your-group.com/public_html/view-me.html
```

The text of the second marker contains:

```
END EXTERNAL
```

The markers are not printed when the FrameMaker document is printed. WebMaker converts the text delimited by the markers into HTML hypertext links.

3.9 Special characters

WebMaker 3.0 converts special characters to their appropriate HTML encoding. Note that there are some special characters that FrameMaker users can enter but which have no encoding in standard HTML 3.2. For example, the

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?

RapidRules does not by default use the library files that include graphic buttons in the navigation panel. To use the library files that do use graphic buttons, edit the WML file in WebMaker and replace the nodes file as follows:

If your file includes **`nodes.wml`**, replace it with **`nodesB.wml`**.

If your file includes **`nodesTI.wml`**, replace it with **`nodesBTI.wml`**.

Then you need to make sure that the GIF files for the graphic buttons are in the directory where the HTML output files are. See Section 3.5.2, "Graphical navigation buttons".

- Why are my graphics all white or missing in the HTML files?

  If you are using graphics that are imported by reference, make sure that any source graphics files (TIFFs, XWDs, and so on) are in the appropriate directories. For example, if the FrameMaker files import graphics files from the same directory as the FrameMaker files are in, and you place the MIF files in another directory, you must copy the graphics files to the directory where the MIF files are. If your FrameMaker files import graphics from a subdirectory of the FrameMaker files directory, and you put the MIF files in another directory, you need to create a subdirectory in the MIF directory that contains all the graphics files.

  Sometimes your browser may be the culprit. If you previously made a web and were missing the graphics files, then if you view the new web, the browser may display the old images, which it cached. Try reloading the images in your browser.

- Why are my graphics cropped in the HTML files?

  There are two reasons why a graphic might be cropped.

  If the anchored frame surrounding the graphic is not at least as large as the graphic itself, the image will be cropped to the size of the anchored frame. For best conversion results, you should make sure all anchored frames are at least as large as the graphic inside of them. Also note that the anchored frame cannot exceed the size of the FrameMaker page.

  Most graphics convert best when imported into your FrameMaker document at 72 dpi. If you are using the PC version of WebMaker, and your FrameMaker files contain graphics imported at any resolution higher than 72 dpi, the graphics will be scaled down to 72 dpi and then con-

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

Complete WebMaker Options

This chapter describes options in the WebMaker user interface. WebMaker provides one main window and three dialog boxes: the WebMaker Wizard, the RapidRules dialog, and the Make a Web dialog. While you are using WebMaker, you are doing two things:

- Viewing and editing a WML file
- Converting documents to HTML using the open WML file

Note that you can convert documents to HTML using a command line (also called batch mode). See Chapter 6, “Making a Web in Batch Mode”.

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.

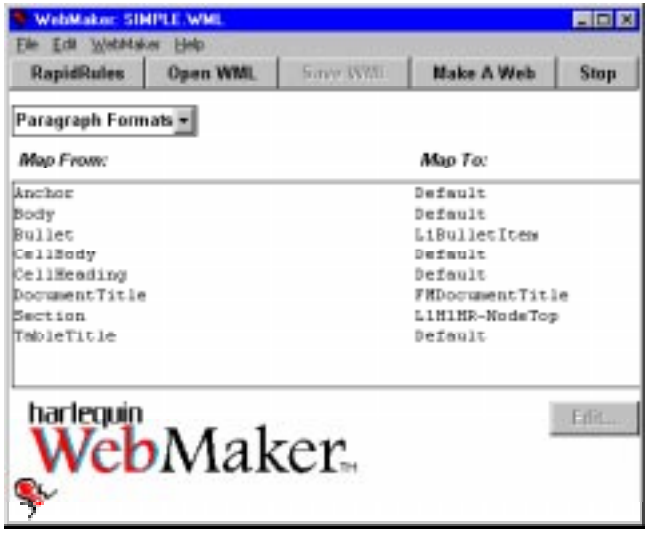

Figure 4.1 WebMaker main window

The main window offers:

- Views of the WML file: The default view is **Paragraph Formats**. You can select other choices from the drop-down list: **Character Formats**, **Format Overrides**, and **WML Libraries**.
- Buttons that accomplish the basic tasks.
- Menu commands

4.2.1 Buttons in the main window

- **RapidRules** opens the RapidRules dialog so that WebMaker will create an initial WML file based on the tags in the MIF file (single document or book) you specify.
- **Open WML** opens an existing WML file.
- **Save WML** saves the current WML to a file.
- **Make a Web** opens the Make a Web dialog so you can convert a document.

- **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:

- **Help > About WebMaker** displays the initial screen, which shows the version of WebMaker you are running. This information is useful if you need help from WebMaker Customer Support.
- **Help > Help** displays information about the batch-mode options to WebMaker.

4.3 RapidRules dialog

To use RapidRules, click **RapidRules** in main window. WebMaker opens the RapidRules dialog box, as shown in Figure 4.2.

Figure 4.2 RapidRules dialog

You specify:

MIF File	Names a MIF file (single document or book) to process. You can specify a MIF file by typing in its pathname or by using the **Browse...** button to specify a file. There is no default.
Node Type	Specifies whether RapidRules should use rules that will result in **Multiple Web Pages** connected by navigation links, or in a **Single Web Page**. The default is **Multiple Web Pages**.

The FrameMaker document you specify must be in MIF format. If you specify a FrameMaker book file, RapidRules and processes each component file in the book. Each file in the book and the book itself must be in MIF format. See Section 2.2, “Write the FrameMaker file to MIF format”, for more information on how to save your FrameMaker files as MIF.

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.

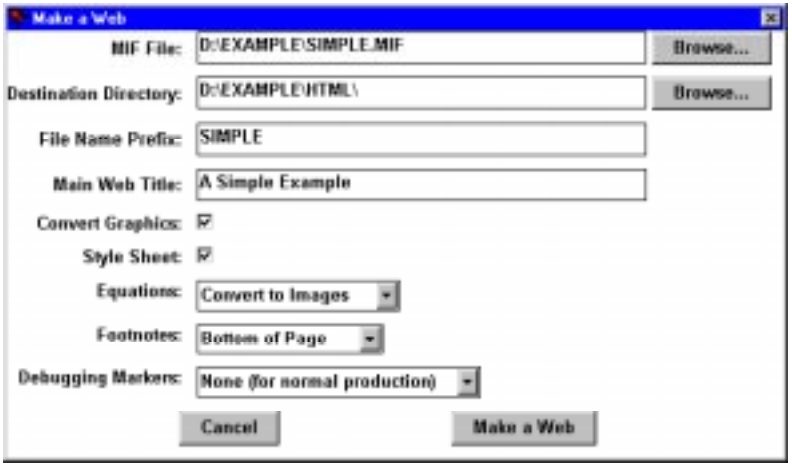

Figure 4.3 Make a Web dialog

Notice now that WebMaker may have filled in the MIF File field for you, if you have used RapidRules or Make a Web in this session. If the default that it chose is incorrect, you can change it.

Now notice that the Destination Directory and File Name Prefix fields are also filled in. The contents of these fields are based on either the name and location of the MIF file in the MIF File field, or, if there is no MIF file, on the name and location of the WML file.

If no fields are filled in, or if you wish to specify other files and prefix, follow these steps:

1. Specify the FrameMaker document (in MIF format) that you want to convert to a web.
2. Specify the destination directory. This is the directory where the HTML files should be created. If you type in a directory that does not exist, WebMaker will create it for you.
3. Specify the filename prefix for the HTML files.

   If you enter a prefix such as **`sim`** (which is appropriate for the Simple Example document), the HTML files are named as follows: **`sim-1.html`**, **`sim-2.html`**, and so on. Note that on platforms where filenames have a

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

already been converted, and you are sure that the resulting HTML files will not have different filenames than they did in the previous conversion.

4. Choose an Equations option. You can choose to have WebMaker convert FrameMaker equations into GIF files or HTML markup. Note that more browsers support the display of GIF files than the display of the HTML equation markup.

5. Choose a Footnote option. The Bottom of Page option puts footnotes at the bottom of the web page. The All on One Page options puts all the footnotes in a single web page. The New Page per Note options puts each footnote in its own web page.

6. Choose a Style Sheet option. If checked, WebMaker creates a **`.css`** file and inserts CSS-related markup into the HTML files. If unchecked, WebMaker does not create a **`.css`** file and does not insert CSS-related markup into the HTML files. For more information, see Chapter 5, “Cascading Style Sheets”. Additional CSS-related options are available as command-line options; see Section 6.2.3, “Options for style sheets”.

7. Choose a Debugging Markers option. The None option is for normal processing and is what you should normally choose. For information, see Section 8.2, “Debugging markers”.

8. Click on **Make a Web**. WebMaker provides a progress bar that gives you an idea of where it is in the process of making a web.

5

Cascading Style Sheets

WebMaker supports *Cascading Style Sheets* (CSS), a major new feature of HTML. For the specification of cascading style sheets, see **`http://www.w3.org/publ/WWW/TR/`**. CSS is supported by Microsoft's Internet Explorer, Netscape Communicator 4.0 (now available in a pre-release version), and W3C's Arena and Amaya browsers.

5.1 Overview of CSS

The purpose of CSS is to allow authors and readers to have greater control over the appearance of HTML documents in browsers. An HTML file can use a style sheet to control the formatting of the document. A *style sheet* is much like a FrameMaker template; it can specify the fonts, the font sizes, the color, margins, indenting, line height, and so on. A FrameMaker template exists within the FrameMaker document, in its master pages, reference pages, paragraph and character catalogs. A cascading style sheet can exist as a separate file outside the HTML file; the HTML file references the external style sheet. The file extension of style sheets is **`.css`**.

Style sheets are combined; the formatting of an HTML document is the result of combining all the style sheets in effect. For example, a document can reference two or more external style sheets. The HTML document can also contain style commands within it; these style commands are combined with all the

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.

5.2.2 Browser problem with vertical margins

Microsoft's Internet Explorer 3.0 browser does not implement vertical margins completely. In particular:

- Bottom margins are not implemented at all.
- Paragraphs have a built-in margin that is not overridden by the CSS margins.
- Vertical margins do not collapse according to the CSS specification.

The problems listed above are fixed in the 4.0 release of Internet Explorer.

WebMaker 3.0 generates CSS instructions that adhere to the specification.

5.2.3 Using an existing style sheet

Note that WML supports CSS by writing the **`class`** attribute with a reasonable default and several options for overriding the default. There are two reasons for overriding the default. The first reason is for people who already have a cascading style sheet with class names that do not map directly to the FrameMaker paragraph tags; those people can use the **`class`** attribute to specify the class names they use in their style sheet. The second reason lies in the fact that FrameMaker tags are case-sensitive but CSS class names are case-insensitive. Thus, if people have a FrameMaker template with two paragraph tag names that differ only in case, they can use the **`class`** attribute to distinguish between them by creating different names. See Section 12.4, "WML support for CSS".

5.3 Hand-crafting the CSS file

The style sheet that WebMaker creates is a reflection of the formatting specified in the FrameMaker document. You might decide that you want the HTML document to have a different appearance than the FrameMaker document. You might want to use different fonts, different sizes, different colors, or add a background color that was not in the FrameMaker document.

You can edit the style sheet created by WebMaker in an ordinary text editor. You will need to become familiar with the rules and syntax of CSS.

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

Making a Web in Batch Mode

When you convert a document again or convert other documents with an existing WML file, you can make a web in batch mode. *Batch mode* means that you give a command to run the conversion, and you do not use the WebMaker user interface. Batch mode is available on the UNIX and Windows 95 platforms, but not on the Macintosh.

The **webmaker** command line has a help mode that displays all the options. To get help, specify one of the following:

```
webmaker -help
```

6.1 Syntax for running WebMaker interactively

To start the WebMaker graphical user interface:

```
webmaker [WML-files]
```

If no *WML-files* are specified, WebMaker opens a window for creating a new WML file. For example:

```
webmaker
```

If you specify one or more *WML-files*, WebMaker opens a window for editing each WML file. For example:

```
webmaker guide.wml
```

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.

-RR ***WML-file*** Behaves like the **-R** option, but will overwrite the *WML-file*, if it exists.

6.2.2 General conversion options

You may specify any of the following *batch-options*:

-single Tells RapidRules to suggest rules for single-page output. The default is multi-page output. The **-single** option is used with **-r**, **-R**, or **-RR**, but not with **-c**.

-t "*title***"** Use *title* as the HTML **TITLE** of the first web page. Take care to quote special characters such as quotation marks and spaces to avoid confusion in the command shell. For example:

```
webmaker -t "My document" ...
webmaker -t 'This is "my" document' ...
```

-w *directory* Put generated HTML files in *directory*. The default is the current directory.

-l *filename* (That's a lowercase L.) Specify a file *filename* for the log. The default directory is the output directory.

-n *name* Prefix of all generated HTML files. The default is the base name of the *document* file.

-f b Write footnotes at the bottom of the HTML page. This is the default treatment for footnotes.

-f o Write footnotes on one separate HTML page.

-f i Write footnotes on individual HTML pages.

-e h Write equations in HTML 3 tags.

-e c Convert equations to GIF graphics.

-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.

-nostyles Do not generate a style sheet and do not insert CSS markup into the HTML files.

-linkstyles ***name*** Create a link on each generated HTML page to the specified style sheet instead of the usual link to the automatically generated style sheet; also do insert CSS markup into the HTML files. Note that WebMaker automatically appends the **.css** extension to the name you provide, so do not provide a name with a **.css** extension. This option is useful in two cases. The first case is for people who already have a style sheet; you can use this option to link to it within the HTML. The second case is for people who want to customize some aspects of the style sheet generated by WebMaker. In this case, you can have the HTML pages link to the customized style sheet, which can in turn link to the automatically generated style sheet that WebMaker produced.

To generate a style sheet and link automatically, specify no style sheet options:

```
webmaker ... mydoc.MIF                    (generates mydoc.css)
```

To specify the name of the generated style sheet, use the **-styles** option:

```
webmaker -styles docstyles ...            (generates docstyles.css)
```

To generate a link on all Web pages to an external style sheet named **localstyles.css** and not to the generated one:

```
webmaker -linkstyles localstyles ...
```

To skip generating a style sheet based on the FrameMaker document:

```
webmaker -nostyles ...
```

To skip generating a document specific style sheet, and generate a link to an external style sheet:

```
webmaker -nostyles -linkstyles localstyles ...
```

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

FrameMaker Style Recommendations

To make a FrameMaker document easy to convert to HTML, you need to use certain style guidelines within the FrameMaker document. The guidelines are:

- Use paragraph and character tags consistently.
- Put graphics in anchored frames.
- Use FrameMaker cross references to make any references to other parts of the document.
- Use FrameMaker's autonumbering for text that needs to be numbered.
- Use heading levels consistently.
- Ensure that master pages contain only page layout information.
- For documents with more than one text flow, ensure that the text flows are in the order that WebMaker expects.
- Consider imported graphics resolution.
- Attach anchored frames to their own paragraphs.

The first five style guidelines are generally considered good FrameMaker style. The last four style guidelines reflect how WebMaker works.

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.

The reason that graphics must be in anchored frames is rooted in HTML. HTML does not specify positional information, except for a sequential order in which objects are to be displayed. Frames that are not anchored to a point in the text are positioned absolutely on a FrameMaker page, and there is no indication in the MIF of their position relative to the text. Thus, it is impossible to reliably position the graphical contents of an unanchored frame in the converted web of WWW files. Because of this, WebMaker ignores all objects not belonging to a Text Column. The only way to record the relative order of graphics in a FrameMaker document is to put graphics in anchored frames.

Anchored frames should also be placed in their own paragraphs in the FrameMaker document; see Section 7.3, “Attach anchored frames to their own paragraphs” for more information.

7.3 Attach anchored frames to their own paragraphs

When inserting anchored frames in your FrameMaker document, you should anchor them to a separate paragraph. When WebMaker processes anchored frames into HTML, it attaches the resulting inline GIF image to the bottom of the preceding paragraph. The bottom of the GIF is aligned with the bottom of the paragraph, causing the image to extend upwards into the paragraph. If the anchored frame is attached to a text paragraph, the result can be blank space between lines of the paragraph. Attaching the anchored frame to its own paragraph ensures that the image will appear in the proper place, separated from the text paragraph, in the HTML file.

You should always put your graphics in anchored frames; see Section 7.2, “Put graphics in anchored frames” for more information.

7.4 Use cross references

Generally, you want any cross reference you make in the FrameMaker document to work smoothly in the converted web of HTML files. For example, assume that your document has several chapters, and each chapter converts to a separate node in the web. If your document contains cross references from one chapter to another, you want the WWW user to be able to click on the cross reference to get to the chapter that is referenced. WebMaker can convert

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.

7.6 Use heading levels consistently

Many FrameMaker documents have a logical hierarchical structure, even though FrameMaker does not enforce any kind of structure. For example, this document is organized with paragraph tags such as: Chapter, 1Heading, and 2Heading. By convention, the writers know that a 2Heading is subordinate to a 1Heading, which is subordinate to a Chapter, and the writers enforce that structure in the document. HTML is hierarchical, in contrast to FrameMaker. Thus, when you convert FrameMaker documents to HTML, it is helpful if your FrameMaker document already has a hierarchical structure.

Here is an example of a document whose paragraphs use heading levels consistently:

```
Chapter
  1Heading
    2Heading
    2Heading
  1Heading
  1Heading
    2Heading
Chapter
```

Here is an example of a document whose paragraphs use heading levels inconsistently:

```
Chapter
    2Heading
  1Heading
```

7.7 Check information within master and reference pages

WebMaker ignores all information contained in master pages and reference pages. Thus, you should ensure that you do not put in master pages or references pages any text or graphics that you want to appear in the HTML document.

7.8 Check the order of text flows

If you have documents that have more than one text flow, it is important to understand how WebMaker handles text flows. In FrameMaker, text flows connect text columns. WebMaker converts text flows sequentially. The order is

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

the page number, such as: See Section 7.10, “Consider cross-reference formats”.

If you want to have different cross-reference formats for the printed document and the web document, you can have two FrameMaker templates, one for paper and one for the web. You can work within the for-paper template, and just before you convert to HTML, import the formats (cross-reference formats only) from the for-web template.

8

Troubleshooting

This chapter describes two ways to get more information from WebMaker (the log window and debugging markers). It also describes the kinds of problems that can occur, and recommends ways to approach those problems.

8.1 The log window

If you notice a problem, it is helpful to open the log window by choosing **`WebMaker > Log`** in the main window. The log might contain a useful message that explains the problem. Even if the message is not helpful to you, it could be helpful to Harlequin support staff when they investigate the problem.

8.2 Debugging markers

WebMaker has a Debugging Markers feature in the "Make a Web" dialog. This feature inserts text in the HTML to indicate which WML rule is being used for each paragraph. The text is visible when you view the HTML files in your browser.

You can choose from three debugging levels in the "Make a Web" dialog box: None, Full, or Summary. The None option is for normal processing; it does not insert any text. The Full option inserts five kinds of markers, at the beginning of each WML-triggered paragraph:

<<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.

These problems are unlikely if you simply include the library WML files, and map your paragraphs to the predefined rules. However, if you write new WML rules or modify existing ones, it is always possible to introduce an error. WML is a programming language, and any WML code you write could be either correct or incorrect. When you open a WML file, WebMaker parses the WML. If WebMaker discovers a problem in the WML file, it tries to give an information message in the log or in a notification window.

- Problems with a specific browser.

  If your output does not look right on one browser, it is sometimes helpful to view it with a different browser. One example is that older versions of Mosaic do not handle HTML tables. If you use a newer version of Mosaic or another browser, the tables appear correctly.

- Problems with WebMaker itself.

  If you find a problem with WebMaker itself (such as WebMaker crashing), please contact WebMaker support by electronic mail:

  `webmaker-support@harlequin.com`

  Provide the following information: the version you are running (choose **WebMaker > About WebMaker** in the main window), any information from the log file (choose **WebMaker > Log** in the main window), and a complete description of what you were doing when the problem occurred. Please fill out the customer support form included with WebMaker. Please be sure the information you give is as complete as possible. The more information you can provide, the more we can help you solve your WebMaker problems.

9

Overview of WML Files

WebMaker operates on WebMaker Language (WML) files. A WML file is a file that describes how a set of FrameMaker paragraph and character tags are to be translated when generating a web of documents, and how to divide the FrameMaker document into separate HTML files (also called nodes). You either create a new WML file (with **File > New**) or open an existing one (with **File > Open**). When you make changes with WebMaker, and then choose **File > Save**, WebMaker saves those changes in the WML file.

WebMaker offers a convenient user interface for viewing and editing WML files. However, a WML file is a plain text file, so you can also use a text editor to view or modify it.

Each WML file must conform to the rules and syntax of the WebMaker Language, which is essentially a programming language. See Section 12, “WebMaker Language (WML)”.

This chapter gives an overview of all the elements that a WML file can contain.

9.1 Two kinds of WebMaker users

WebMaker supports two kinds of users: the library user, and the WML configurer.

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
```

On Windows you might see:

```
c:\webmaker\lib\nodesTI.wml
c:\webmaker\lib\headings.wml
c:\webmaker\lib\normals.wml
c:\webmaker\lib\contents.wml
c:\webmaker\lib\lists.wml
```

On the Macintosh you might see:

```
Banana:Applications:WebMaker:lib:nodesTI.wml
Banana:Applications:WebMaker:lib:headings.wml
Banana:Applications:WebMaker:lib:normals.wml
Banana:Applications:WebMaker:lib:contents.wml
Banana:Applications:WebMaker:lib:lists.wml
```

You can view the WML statement for an include file by looking at the WML file in an ordinary text editor. For example:

```
INCLUDE <nodesTI.wml>
```

For more information about include files, see Section 12.2, “Include files”.

9.2.1 Relative pathnames of include files

You may specify an include file with 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.

Relative pathnames have an advantage when using WebMaker on multiple platforms: you can conveniently move a directory structure from one platform to another, and the include files will continue to work (as long as the organization of the directories is preserved).

Absolute pathnames have the advantage of sharing; many WML files can include the same files, such as library WML files.

See Section 12.2, “Include files” for information on the syntax of specifying include files in a WML file.

9.2.2 Organizing WML files

One useful technique for organizing WML files is to use or create these kinds of WML files:

- 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

definition of a uses rule named **Bullet**, which uses a primary rule named **L1BulletItem**:

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

9.3.2 Recommended technique for paragraph rules

When you convert a FrameMaker document to HTML, you want to specify how each paragraph in your FrameMaker document should appear in HTML. To do this, you need to have a paragraph rule with the same name as each paragraph tag in your FrameMaker document.

The easiest way to create paragraph rules that correspond to the paragraph tags is to have RapidRules create a WML file for your document. The resulting WML relies exclusively on WML library files for its mappings, which means that every paragraph rule is a uses rule.

The two paragraph rules in Section 9.3.1, "Categories of paragraph rules: primary and uses" are examples of this technique. The primary paragraph rule **L1BulletItem** is defined in the library file **lists.wml**. The uses rule **Bullet** is defined in a WML file created by RapidRules. The Bullet paragraph tag exists in the FrameMaker document.

9.4 Overview of character rules

A character rule maps a FrameMaker character tag to an HTML character highlight. You can see the character rules that are defined in a WML file by selecting the **Character Formats** view in the main window. You can click **Edit** to specify another HTML mapping for a character tag. The complete list of HTML highlights is:

```
Big, Bold, Cite, Code, Definition, Emphasise, Italic, Keyboard,
None, Sample, Small, Strikethrough, Strong, Subscript,
Superscript, Teletype, Underline, Variable
```

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

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”.

9.7 Overview of node rules

A WML file can contain definitions of node rules. A node rule specifies the appearance of web pages that use that node rule, such as the header and footer of the web page. A node rule is a template for the nodes. The header and footer typically contain navigation panels, which provide hypertext links for navigating your web of documents. A node rule is used by one or more paragraph rules. Each paragraph that starts a new node must use a node rule.

You can view the WML statement for a node rule by viewing the WML file in an ordinary text editor. For example:

```
NODE TOCNode
{
  TITLE headingtext()
  HEADER
  {
    @NavPanel=concatenate(
              button("[Next] ", filename(next), "[Next]" ),
              button("[Previous] ", filename(previous),
                                                  "[Previous]"),
              button("[Top] ", filename(top), "[Top] " ),
              button("[Index] ", filename(INDEX), "[Index] " )
                        );
    write(*,paragraph(@NavPanel));
    write(*,paragraph(maintitle()));
  }
  FOOTER
  {
    write(*,toc(4,global));
    write(*,hrule());
    write(*,address(concatenate(maintitle()," - ",date())));
    write(*,paragraph(@NavPanel));
  }
}
```

For completeness, we show a paragraph rule that uses this node rule. The line **`NEWNODE TOCNode`** is the place where this paragraph rule specifies that it starts a new node, and that the node is defined by the **`TOCNode`** rule.

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

10

Modifying Conversion Rules, Headers, and Footers

There are two cases in which you need to edit WML files directly:

1. When you have experimented with the conversion rules available in the libraries, and you need a behavior that no predefined rule provides. In this case, you can create a new WML rule. See Section 10.1, “Creating new WML conversion rules”.
2. When you want to customize the appearance of the headers or footers of the HTML pages. See Section 10.2, “Changing headers and footers”.

WebMaker Language (WML) is a macro language documented fully in Chapter 12, “WebMaker Language (WML)”. It is possible to read the reference documentation of WML and create WML files from scratch. However, it is much easier to copy the definition of an existing rule and modify it. Typically, you can find a rule that does almost what you want, copy that rule, and modify the copy of that rule to do exactly what you want. This chapter describes how to do so.

10.1 Creating new WML conversion rules

The process that we recommend for creating new conversion rules is:

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:

```
# QuotedText specifies that text be displayed as a BLOCKQUOTE,
# separated from text above and below by a blank line.
PARAGRAPH "QuotedText" TYPE Normal
{
  CONTEXT Quote
  ACTIONS
  {
    write(*,paragraph(text()));
  }
}
```

We now edit the copy of the rule to have a new name, **QuotedTextItalic**, to have a revised comment that describes this new rule, and to use the **italic** WML function. The definition of the **QuotedTextItalic** rule is:

```
# QuotedTextItalic is like QuotedText, but displays in italics.
PARAGRAPH "QuotedTextItalic" TYPE Normal
{
  CONTEXT Quote
  ACTIONS
  {
    write(*,paragraph(italic(text())));
  }
}
```

10.1.2 Example: Creating a rule for warnings

In this example, we want to modify the **QuotedText** rule to be used for warning paragraphs. Thus, we copy the **QuotedText** rule, rename it to **WarningBold**, edit the comment to describe the new rule, use the **concatenate** WML function to write out the word Warning:, and use the **strong** WML function to make the word Warning: appear in a strong character font. The definition of the **WarningBold** 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())));
  }
}
```

10.2 Changing headers and footers

You can consider each web page as having a header and a footer, much like a printed page. The header is any information that appears at the top of the web page, before the content. The footer is any information that appears at the bottom of the web page, after the content. Typically, the headers and footers include navigation buttons, the main title of the web document, and the date of update. In the predefined rules, the phrase "Generated by Harlequin WebMaker" and a link to the Harlequin home page appears in the footer.

You might want to customize the headers or footers. For example, you might want the footer to contain information about your company, to display your company's logo, to contain a copyright notice, and to remove the phrase "Generated by Harlequin WebMaker".

The headers and footers are controlled by node rules. Node rules are in the predefined files **`nodesB.wml`**, **`nodesTI.wml`**, **`nodesBTI.wml`**, and **`nodejava.wml`**. Each of these files contains a set of node rules. The node rules are not mapped directly to paragraph tags; instead, paragraph rules are defined to use node rules. For example, the **`L1H1NodeTop`** paragraph rule uses the **`LevelOne`** node rule.

We recommend a slightly different procedure for editing node rules than the procedure for editing paragraph conversion rules. For conversion rules, we copy a few rules to another file and modify those rules. For node rules, we recommend this procedure:

1. Identify the predefined node rules file that most closely matches the conversion behavior you want to achieve. For example, we identify the **`nodesBTI.wml`** file.
2. Copy the entire file to a new filename, such as **`custom-nodes.wml`**.
3. Edit the node rules in **`custom-nodes.wml`** to change the appearance of the headers and footers, as desired. Do not change the names of the node rules. They must keep the same names because the paragraph rules are defined to use node rules of these names.
4. As an initial test of whether you have written correct WML syntax, open the **`custom-nodes.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-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(*,paragraph(@NavPanel));
    write(*,paragraph(maintitle()));
  }
  FOOTER
  {
    write(*,toc(1,local));
    write(*,hrule());
    write(*,address(concatenate(maintitle()," - ",date())));
    write(*,paragraph(@NavPanel));
    @Hqn=concatenate(
                     "Generated with ",
         button("Harlequin WebMaker","http://www.harlequin.com/
webmaker")
                    );
    write(*,paragraph(@Hqn));
  }

}
```

Figure 10.1 Original definition of LevelOne node rule.

```
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).

10.3 Using different graphics for navigation buttons

You might have graphics that you want to use for the navigation buttons instead of using WebMaker's graphics for Next, Previous, and so on. It is a simple matter to substitute your own graphics. In the directory containing the HTML files, copy your graphics file into the file **`next.gif`**. Similarly, copy your files into the files named **`prev.gif`**, **`content.gif`**, **`index.gif`**, **`top.gif`**, and **`up.gif`**. Note also, that you will also need graphical buttons named **`nextg.gif`**, **`prevg.gif`**, and so on, which are greyed-out versions of the Next, Previous, and other buttons. Those gif files are used when the Next, Previous, and other buttons are disabled.

11

WML Library Rules Reference

These rules are defined in the library of WML files provided with WebMaker.

11.1 Overview of library of WML files

The WML library contains the following files in the **lib** directory of your WebMaker installed directory:

nodes.wml Contains node rules for a first page, level one, and level two and lower layout templates. 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 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 file uses text rather than images for navigation buttons.

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 Node libraries

Node rules specify the web page templates for a web. The template includes the title, header, and footer for each page. The WebMaker library includes several WML files for creating nodes. You include one node library file in your document to define the templates for web pages.

The node rules are used indirectly. Header rules reference the specific node rule they need if they produce new nodes. A first level header will use a level one node rule, an index heading will use an index node rule, and so on. Different pages require different navigation hyperlinks. These links appear in the headers and footers of a page and their appearance is defined in the node rules. Each node library includes several node rules because most web documents call for several kinds of web pages. Some also include specialized paragraph rules for indexes and tables of contents.

One of the node libraries must be included whenever you wish to use any node-generating rules from the headings library.

11.2.1 The nodes.wml file

The **`nodes.wml`** file contains definitions of simple node templates with text navigation links for a web with no external table of contents or index.

11.2.2 The nodesB.wml file

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

```
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.

11.2.3 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:

```
AboutBox.class     TOCType.class
HTMLTag.class      URLButton.class
NavParser.class    URLList.class
TOCPanel.class     WMNavigator.class
```

11.3 Normals

Normals rules specify common body text display.

`Default` *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 **`NumberedBody`**.

`FixedWidthText` *Normal*

Specifies display in a fixed-width font.

`FMDocumentTitle` *Normal*

Specifies that text be displayed as a level 1 header followed by a rule.

`Ignore` *Normal*

Specifies that the text should neither be written to HTML nor displayed.

`InListNumbered` *Normal*

Specifies a paragraph preceded by a number and a space with the same text indentation as the preceding paragraph. The number is the character or characters created by FrameMaker autonumbering for this item.

`InListSimple` *Normal*

Specifies a paragraph with the same text indentation as the preceding paragraph.

NumberedBody *Normal*

Specifies a simple body paragraph preceded by a number and a space. The number is the character or characters created by FrameMaker auto-numbering 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.

	The numbering stream starts over at 1 after any non-list item except InListSimple or InListNumbered.
Autonumber	Specifies an unordered list item that displays the FrameMaker autonumber for the paragraph. Levels specify amount of indentation.
Plain	Specifies a plain list item, without bullet or number. Levels specify amount of indentation.
Level	Specifies the hierarchy level. Level 2 items are subtopics of level 1 items and so on. Hierarchy displays with greater indentation for higher levels.

`L1GlossaryItem-1` *List*

Term and definition are plain text.

`L1GlossaryItem-2` *List*

Term is bold; definition is plain text.

`L1GlossaryItem-3` *List*

Term and definition are plain text. The first two tab-delimited columns of the source text are used as the term. The third and following columns are used as definition.

`L1GlossaryItem-4` *List*

Term is bold; definition is plain text. The first two tab-delimited columns of the source text are used as the term. The third and following columns are used as definition.

`L1GlossaryItem-5` *List*

Term is a FrameMaker autonumber in bold; definition is plain text.

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.

L2PlainItem *List*

Specifies level 2 plain list item.

L3PlainItem *List*

Specifies level 3 plain list item.

11.5 Headings

Headings rules create the hierarchical structure of the web document. The division of a FrameMaker document into separate web pages is controlled by headings rules. The template for those separate pages is specified by node rules referenced in headings rules. You must include one of the node rules files (**nodes.wml**, **nodesTI.wml**, **nodesB.wml**, **nodesBTI.wml**) from the library to use any of the headings that create nodes.

The file **headings.wml** provides two kinds of headings: *special headings*, for creating an index or table of contents; and *body headings*, for regular text headings.

11.5.1 Special headings

Special headings rules are used once in a document or book to produce a table of contents or index.

ExtTOCHeading-TocNode *Heading*

Level 1 in the hierarchy.

Heading 1 display size.

HR horizontal rule.

External table of contents node. This rule should be used only once for a document or book to produce an external table of contents node. A tag mapped to this rule, such as WWW-TOC, should appear only once in the FrameMaker document or book.

Include **nodesTI.wml** or **nodesBTI.wml** from the library to create an external table of contents.

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.

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.

IndexHeadingS-IndexNodeS *Heading*

Level 1 in the hierarchy.

Heading 1 display size.

Index node, simple. Index entries are sorted alphabetically and displayed without headers dividing the entries into sections.

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.

11.5.2 Body headings

The body heading rules specify up to four characteristics.

Level	From 1 to 4, specifies the hierarchy level. Level 2 headings are subtopics of level 1 headings and so on. Table of contents uses the value of level for the indentation of the table of contents entry for a heading.
Heading	From 1 to 4, specifies the size of characters used to display the text. H1 is the largest. These correspond to the HTML tags **H1**, **H2**, **H3**, and **H4**.
HR	Specifies a horizontal rule below the text. Corresponds to HTML tag **HR**.
Node	Specifies a node rule for headers that produce a new node.

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
L4H1HR-NodeLower	4	<H1>	Yes	Yes	Next, Previous, Top, Up
L4H1-NodeLower	4	<H1>	No	Yes	Next, Previous, Top, Up
L4H2-NodeLower	4	<H2>	No	Yes	Next, Previous, Top, Up
L4H3-NodeLower	4	<H3>	No	Yes	Next, Previous, Top, Up
L4H4-NodeLower	4	<H4>	No	Yes	Next, Previous, Top, Up
L1H1	1	<H1>	No	No	
L1H1HR	1	<H1>	Yes	No	
L2H1	2	<H1>	No	No	
L2H1HR	2	<H1>	Yes	No	
L2H2	2	<H2>	No	No	
L2H2HR	2	<H2>	Yes	No	
L3H1	3	<H1>	No	No	
L3H1HR	3	<H1>	Yes	No	
L3H2	3	<H2>	No	No	
L3H2HR	3	<H2>	Yes	No	

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

the line (on the assumption that it is the page number). If your table of contents has a different format, you can define custom TOC rules.

TOC-Entry-1 *List*

Specifies a table of contents entry with level 1 of indentation.

TOC-Entry-2 *List*

Specifies a table of contents entry with level 2 of indentation.

TOC-Entry-3 *List*

Specifies a table of contents entry with level 3 of indentation.

TOC-Entry-4 *List*

Specifies a table of contents entry with level 4 of indentation.

AML-Entry-1 *List*

Specifies an alphabetic marker list entry with level 1 of indentation.

APL-Entry-1 *List*

Specifies an alphabetic paragraph list entry with level 1 of indentation.

LOF-Entry-1 *List*

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

LOM-Entry-1 *List*

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

LOP-Entry-1 *List*

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

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

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

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

11.6.2 The list2.wml file

The file **`list2.wml`** provides all of the list rules in **`lists.wml`** (described in Section 11.4, “Lists”) plus the following list rules:

`L4BulletItem`	**`L4AutonumberItem`**
`L5BulletItem`	**`L5AutonumberItem`**
`L6BulletItem`	**`L6AutonumberItem`**
`L4NumberItem`	**`L4PlainItem`**
`L5NumberItem`	**`L5PlainItem`**
`L6NumberItem`	**`L6PlainItem`**

If you use **`list2.wml`**, you do not need to use **`lists.wml`**.

12

WebMaker Language (WML)

For WebMaker to map an unstructured sequence of paragraph formats into the hierarchical and recursively structured HTML environment, some extra information is necessary. The rules for mapping a FrameMaker document into HTML are specified in the WebMaker Language, WML.

WML is intended to be easy to read and write while still providing full flexibility to map onto any legal HTML construct. HTML elements are referred to logically and never by raw HTML. WML provides a set of predefined functions, and supports the ability to define variables to facilitate the configuration.

Warning: Full checking of the HTML output for a given configuration is not implemented. It is possible to configure WebMaker to produce incorrect HTML.

Before reading the reference information in this chapter, you should be familiar with the material covered in Chapter 9, “Overview of WML Files”.

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.

12.1.4 Execution model

WebMaker translates a FrameMaker document paragraph by paragraph. This process is outlined simply by the following pseudo-code:

```
FOREACH Paragraph in FrameMaker document
   READ the paragraph
   IDENTIFY the FrameMaker tag
   IF conversion rules for that tag are specified in
configuration THEN
      IF a new HTML node is to be created THEN
         WRITE the footer of the current HTML node
         CLOSE the current HTML node
         CREATE a new HTML node and make this the current node
         WRITE the title and the header of the current node
      ENDIF
      CONVERT the paragraph to HTML
      EXECUTE the conversion actions specified in the
configuration
   ENDIF
END
```

12.1.5 Comments

You can provide comments in a WML file. To provide a comment, use the **#** character at the beginning of a line. For example:

```
# Bulleted lists
```

12.2 Include files

You can define include files. Each include file is treated as if all the definitions in it appeared in the WML file you are editing. The general syntax is:

INCLUDE <*filename*>

INCLUDE "*filename*"

The angle bracket syntax (**<*filename*>**) allows you to include the standard library WML files without being required to specify the directory they are in. When WebMaker sees an include file named by **<*filename*>**, it looks for the file in the **lib** subdirectory of the directory in which WebMaker is installed. So, for example, if you have installed WebMaker for the PC in **C:\webmaker**, WebMaker will look for **<nodes.wml>** in **C:\webmaker\lib\nodes.wml**.

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:

```
INCLUDE <nodesBTI.wml>
INCLUDE <headings.wml>
INCLUDE <normals.wml>
INCLUDE <lists.wml>
INCLUDE <contents.wml>
```

You can combine standard library WML files and other files. You can, for example, use the double quotes syntax to include files that are in directories other than WebMaker's **lib** directory:

```
INCLUDE <normals.wml>
INCLUDE <headings.wml>
INCLUDE "C:\office\memos.wml"
INCLUDE "personal.wml"
```

12.3 User-defined variables

You can define global variables that you can subsequently use in the definitions of node rules, paragraph rules, and character rules.

Variables are defined with:

VARIABLE ***@string***

The WebMaker keyword **VARIABLE** defines a variable. The **VARIABLE** keyword must be followed by a character string that identifies the variable. This identifier must start with the **@** character, and must be followed by an alphanumeric character. The rest of the identifier may be made up of any combination of alphanumeric and the **-** and _ characters.

```
VARIABLE @initialsdate
...
Node FirstPage
{ ...
  HEADER {
            @initialsdate=concatenate("MR - ", date());
            write(*,address(@initialsdate));
...
```

12.4 WML support for CSS

WML supports Cascading Style Sheets. In particular, you can specify the class attribute of HTML elements in a variety of ways.

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 **NAVPANEL**. 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"}
   ```

3. **The `class` keyword argument to HTML markup functions.** All the WML functions that produce HTML markup now take an optional class keyword argument. If it is provided, it overrides the defaults described in **1** and **2**. For example, you can create a rule that provides the class keyword argument to the **`listitem`** function, such as this:

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

   Now, if you map your Bullet paragraph tag to the **`Standard-L1BulletItem`** rule, the HTML will appear like this:

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

Note that WML supports CSS by writing the **`class`** attribute with a reasonable default and options for overriding the default. There are two reasons for overriding the default. The first reason is for people who already have a cascading style sheet with class names that do not map directly to the FrameMaker paragraph tags; those people can use the **`class`** attribute to specify the class names they use in their style sheet. The second reason lies in the fact that FrameMaker tags are case-sensitive but CSS class names are case-insensitive. Thus, if people have a FrameMaker template with two paragraph tag names that differ only in case, they can use the **`class`** attribute to distinguish between them by creating different names.

12.5 Node rules

Node rules are the equivalent of FrameMaker master pages. Each node rule describes an HTML page type independently of its contents. Node rules typically define the navigation panels.

To use a node rule, you associate it with a node-triggering paragraph tag, as described in Section 12.6, “Primary paragraph rules”. You can use a node rule in more than one node-triggering paragraph tag. It is useful to have different

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”.

An example of a node rule is:

```
# Define the entry page of the generated web.
#
NODE FirstPage
{
  TITLE maintitle()
  HEADER
  {
    @NavPanel=concatenate(
              button("[Next] ",filename(next),"[Next] ")
                          );
    write(*,paragraph(@NavPanel));
  }
  FOOTER
  {
    write(*,toc(1,local));
    write(*,hrule());
    write(*,address(concatenate(maintitle()," - ",date())));
    write(*,paragraph(@NavPanel));
  }
}
```

12.5.1 Specification of attributes of a node rule

The node attributes include:

```
TITLE  "the node HTML title"
CLASS  "class-name"
```

where:

TITLE must be followed by a character string to be used as the HTML **TITLE** of the node.

CLASS must be followed by a character string, which can be enclosed in double quotation marks or not. This string is the name of the class of all HTML tags written by this node rule, which includes only the HTML generated by the WML commands in the **HEADER** and **FOOTER**. This is an optional attribute. If you use it, all the HTML tags written by commands in the **HEADER** and **FOOTER** will have the same class name (unless the individual WML

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:

1. An HTML **TITLE** that is the same as the heading paragraph that triggered the node.
2. A navigation panel containing links to next and previous nodes, to appear both at the beginning and the end of the node.
3. The heading of the upper section to appear at the beginning of the node.
4. A local table of contents of the sections that are up to two levels lower, and specified at the end of the node.
5. A signature to appear at the end of the node.

This is done by:

```
NODE BasicSection
{
  TITLE headingtext(current)
  HEADER
    {
      @NavigationPanel=concatenate(
                  button("[Next] ",filename(next)),
                  button("[Prev] ",filename(previous)));
      write(*,paragraph(@NavigationPanel));
      write(*,paragraph(headingtext(up)));
    }
  FOOTER
    {
      write(*,toc(2,local));
      write(*,address(concatenate("MR - ",date())));
      write(*,paragraph(@NavigationPanel));
    }
}
```

12.5.4 The first page of the generated web

The entry page of a generated web is treated as a special case for the following reasons:

- Whether the first paragraph in the FrameMaker document triggers a node or not, a first page must always be created.
- It may not always be possible to determine the HTML **TITLE** for the entry page because:

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.

```
NODE FirstPage
{
  TITLE maintitle()
  HEADER
    {
      @NavigationPanel=concatenate(button("[Next] ",filename(next
)));
      write(*,paragraph(@NavigationPanel));
    }
  FOOTER
    {
      write(*,heading(2,"Table of contents"));
      write(*,toc(1));
      write(*,hrule());
      write(*,address(concatenate("MR - ",date())));
      write(*,paragraph(@NavigationPanel));
    }
}

PARAGRAPH "Title" TYPE Normal
{
  ACTIONS  {
             write(*,heading(1,text()));
             write(*,hrule());
           }
}
```

12.6 Primary paragraph rules

A paragraph rule specifies how the text of FrameMaker paragraphs, identified by their paragraph tag, is to be converted to HTML constructs.

The syntax of primary paragraph rules is:

```
PARAGRAPH "FMparatag" TYPE type
{
 specification of attributes
  ACTIONS {
             statement of conversion action
           }
}
```

where:

"*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**.

`Class` must be followed by a character string, which can be enclosed in double quotation marks or not. This string is the name of the class of all HTML tags written by this paragraph rule. If you do not provide the **`Class`** attribute, the default class name is the name of the FrameMaker paragraph tag that was mapped to this paragraph rule. In most cases, the default is fine. See Section 12.4, “WML support for CSS”.

`FileNameKey` A character string to be used as a file key. This may be later passed as a parameter to the function **`file-name(`**node) to be able to point to the HTML file that is triggered by this paragraph from other places in the generated web. This character string is case-insensitive. For an example see Section 12.9.1, “Generated table of contents”. This option may be specified only when **`NewNode`** is specified.

If more than one node is created with the same file name key, then only the first may be accessed with **`filename(`**node).

Each of the three WebMaker paragraph types has attributes that provide further structural and environmental information needed for the generation of a Web. Each TYPE has a set of attributes pertaining to it:

Heading

`Level` Positive integer, 1 being highest. No lower limit. The **`Level`** must be specified; it has no default.

`NewNode` The name of a previously defined node rule.

List

`Level` Positive integer; 1 being highest, no lower limit. There is no default; this option must be specified.

`Kind` **`Number`**, for an Ordered List.

`Bullet`, for an Unordered List. This is the default.

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());
      }
  }
```

```
PARAGRAPH "Step" TYPE List
  {
    LEVEL 2
    KIND Number
    ACTIONS
      {
        write(*,listitem(paragraph(text())));
      }
  }

PARAGRAPH "Warning" TYPE Normal
  {
    ACTIONS
      {
        write(*,hrule());
        write(*,paragraph(concatenate(
                          image("warning.gif",top),
                          bold(text()))));
        write(*,hrule());
      }
}
```

12.6.4 Default actions for the paragraph rule elements

For **Heading** paragraph rules, if no actions are provided, the default action is:

```
write(*,heading(1,concatenate(number()," ",text())));  }
```

For **List** type paragraph rules, if **Kind Bullet** or **Kind Number** is specified, the default action is:

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

For **List** type paragraph rules, if **Kind Glossary** is specified, the default action is:

```
{
  write(*,glossterm(text(current,1,1)));
  write(*,glossdescription(text(current,2,*)));
}
```

For **Normal** type paragraph rules, if **Context Preformatted** is not specified, the default action is:

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

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:

Big - **Bold** - *Cite* - Code - ***Definition*** - *Emphasise* - *Italic* - `Keyboard` - None - **Sample** - Small - ~~Strikethrough~~ - **Strong** - $_{\text{Subscript}}$ - $^{\text{Superscript}}$ - Teletype - Underline - ***Variable***

The basic syntax of character rules is:

```
CHARACTER { TAG "FMchartag" MAP HTML-highlight> )
```

Here is an example:

```
CHARACTER { TAG "Substitute" MAP Sample }
```

You can optionally provide a class name, as in this syntax:

```
CHARACTER { TAG "FMchartag" MAP HTML-highlight>
            CLASS:"class-name" }
```

The syntax elements of character rules are:

"*FMchartag*" is the tag name of the FrameMaker character tag. Because FrameMaker character tags are case-sensitive, this name must match the case of the character tag.

HTML-highlight

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

"*class-name*" This string is the name of the class of all HTML tags written by this character rule. If you do not provide the **Class** attribute, the default class name is the name of the FrameMaker character tag that was mapped to this character rule. See Section 12.4, “WML support for CSS”.

12.8.2 Syntax of format override rules

WebMaker recognizes the **ANGLE**, **FAMILY**, **VAR** and **WEIGHT** parameters of FrameMaker characters. WML character conversion rules may specify differ-

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}
```

In cases where characters match more than one format override rule, the most detailed rule takes precedence. In the preceding 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 those characters.

12.9 Table of contents in WML

WebMaker can produce a local table of contents, a mapped table of contents, and a generated table of contents. For basic information about these three kinds of table of contents, see Section 3.6, “Table of contents”.

12.9.1 Generated table of contents

We define a *generated table of contents* as one generated by WebMaker, not FrameMaker.

To trigger a generated table of contents, the FrameMaker document must include a single use of a FrameMaker paragraph tag, such as: WWW-TOC. The node type dedicated for this use needs to be specified because:

- The function **toc(**depth, scope) needs to be called with **<depth>** as maximum and **<scope>** as **global**. This is normally not desirable in any other node of the generated web.
- This allows for the specification of a **FILENAMEKEY** for the node of the generated TOC, which in turn provides a mechanism to be able to point to this node from anywhere else in the generated web, such as in the navigation panel of other nodes.

For example, suppose we need to create a generated full table of contents for a FrameMaker document that has four levels of headings, and we would like to be able to point to it at will. We need:

- A paragraph tag, used only once in the FrameMaker document, that is used to trigger the creation of the generated TOC.
- A **FILENAMEKEY** for the HTML file of the generated TOC.
- A node type dedicated for the generated TOC.

This is achieved by the following excerpts of WML:

```
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-

tents button. First, if any paragraph in the document has a tag that is mapped to the rule **ExtTOCHeading-TocNode**, then the HTML file containing that paragraph is the distinguished table of contents. Otherwise, if any paragraphs in the document have tags that are mapped to any of the TOC-Entry rules, then the first HTML file containing a paragraph whose tag was mapped to a TOC-Entry rule is the distinguished table of contents.

Note that the expectation of the library rules is that users will choose either a generated table of contents or a mapped table of contents, but not both.

12.10 List of figures or tables in WML

It is possible to create customized WML that will create a mapped list of figures or a mapped list of tables using the same model as the mapped table of contents. For examples of rules for these kinds of lists, see the **contents.wml** library file.

The first requirement is that the FrameMaker document itself contains a list of figures (or a list of tables). You must be sure that the Generate Hypertext Links option is checked when you create the list.

Next, copy the **contents.wml** library file to another filename, such as **LOF.wml**. You can edit the TOC-Entry rules to be LOF-Entry rules, by changing their names and the value of **FILENAMEKEY** such as:

```
PARAGRAPH "LOF-Entry-1" TYPE List
{
  FILENAMEKEY LOF
  LEVEL 1
  KIND Glossary
  ACTIONS
  {
     write(*,glossterm(bold
            (butlast(concatenate(number()," ",text())))));
  }
}
```

Next, copy either the **nodesTI.wml** or **nodesBTI.wml** file to another filename, such as **nodes-LOF.wml**. You can then add a new button to the navigation panel, such as Figures, which uses the filenamekey LOF. For example, you would modify the **FirstPage** node as follows:

```
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`**

indexoption	May take one of three values: **simple**, **letter**, or **node**. The value **simple** creates a single node index with letter groups separated only by a space. The value **letter** creates a single node index with letter groups separated by letter headings. The value **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).

An index is triggered by the single use of a FrameMaker paragraph tag, such as WWW-IX. A node type dedicated for this use should be specified because:

- The navigation panel of the index node is usually desired to be different than the rest of the web nodes.
- This allows for the specification of a **FILENAMEKEY** for the index node, which in turn provides a mechanism to be able to point to this node from anywhere else in the generated web, such as in the navigation panel of other nodes. This is best explained in the following example.

12.11.1 Example of creating an index

Suppose we need to create an alphabetic index from the FrameMaker markers of type Index, with each letter group separated by a large letter, and with the possibility to be able to point to the index node at will. We need:

- A paragraph tag, used only once in the FrameMaker document, that is to trigger the generation of the index.
- A **FILENAMEKEY** for the HTML file of the index.
- A node type dedicated for the index.

This is achieved by the following excerpts of WML:

```
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

LUV A logical unit variable.

filename A string.

mode **new** or **append**.

Opens the file *filename* with *mode* for writing. Subsequent output to this file must be specified through *LUV*.

For example:

```
openfile(@tocfile,"/home/username/xyzTOC.html",append) --> ""
```

write(*filestream, string1, string2, ..., stringn*)

Parameters

filestream A file stream variable identifier. *filestream* may be:

*****, to write into the current HTML file.
stdout, to write to standard output.
LUV, to write into the file associated with the logical unit variable, *LUV*, that is set by **openfile()** when the file is initially opened.

string1... An unlimited sequence of character strings.

Writes the concatenated sequence of strings followed by a carriage return into the file specified by *filestream*.

For example:

```
openfile(@list-of-figures,"/home/rousseau/figTOC.doc",append);
write(@list-of-figures,text());
```

12.12.2 String manipulation functions

concatenate(*string1,string2, ..., stringn*)

Parameters A sequence of character strings separated by commas.

Returns the sequence of strings as one concatenated string.

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.

index1 the i^{th} piece of tab-delineated text

index2 the j^{th} piece of tab-delineated text

Returns a character string that is the concatenation of the i^{th} to the j^{th} piece of tab-delineated text in the current or previous paragraph.

Note: Paragraphs with tab-delineated text may be considered as an array of text pieces. This function allows for the extraction of any sequence or all of these pieces.

This function has useful defaults for the two index parameters, which are demonstrated in the following examples:

`text()-->` all text of current paragraph

`text(previous)-->` all text of previous paragraph

`text(current,1)-->` first piece of current paragraph

`text(previous,2,4)-->` second to fourth pieces of previous paragraph

`text(current,3,*)-->` from the third piece to the end of current paragraph

`time()`

Returns a string containing the current time.

For example:

```
time() --> "17:37:23"
```

`uppercase(`*`string`*`)`

Parameters *string* is a character string.

Returns *string* in all uppercase.

For example:

```
uppercase("AbCdEf 567") --> "ABCDEF 567"
```

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"
```

12.12.3 HTML node functions

`filename(`*`node`*`)`

Parameters *node* is one of the following HTML node identifiers: **`current`** (this is the default), **`top`**, **`up`**, **`next`**, **`previous`**, or *filenamekey* (the filenamekey for a specified node)

Returns a character string with the filename of the *node* file.

For example:

```
filename(top) --> "document_1.html"
```

`headingnumber(`*`node`*`)`

Parameters *node* is one of the following HTML node identifiers: **`current`** (this is the default), **`up`**, **`next`**, **`previous`**.

Returns a character string with the FrameMaker autonumber of the heading paragraph that triggered *node*. See also **`number(currprev)`** and **`headingtext(node)`**.

For example:

```
headingnumber(next) --> "4.5.3"
```

`headingtext(`*`node`*`)`

Parameters *node* is one of the following HTML node identifiers: **`current`** (this is the default), **`up`**, **`next`**, **`previous`**.

Returns a character string with the text of the FrameMaker paragraph that triggered *node*.

Note: The returned string contains no markup (character highlights, image data, hypertext links). This function is meant to be used to isolate the HTML **`TITLE`** of a node, which does not allow any markup. The same paragraph will be separately translated as a heading according to rules specified for its FrameMaker paragraph tag. See also **`heading(level, string, class:classname)`**.

For example:

```
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)
```

maintitle()

Returns the character string that is the main title of the Web. When the WebMaker user interface is used, this is the value specified in the Make a Web dialog, in the Main Title area. When the **webmaker** command line is used, this is the value specified with the **-t** switch.

This function provides the HTML **TITLE** of the entry page of a translated document.

toc(*depth, scope*)

Parameters

depth	A positive integer
scope	**local** or **global**. The default is **local**.

Returns a string that contains the HTML markup for a table of contents for a node of the given *depth*, starting from the level immediately below current if *scope* is **local**, or starting from the topmost node if *scope* is **global**.

Only paragraph tags declared as **TYPE Heading** are considered as eligible entries in a table of contents.

The *depth* corresponds to the **LEVEL** parameter of paragraphs of **TYPE Heading** only when the *scope* is **global**. For example:

```
toc(3,global)
```

The example above returns the table of contents for all occurrences of paragraphs of **TYPE Heading** that have **LEVEL** attribute values of either **1**, **2**, or **3**.

12.12.4 Class argument to many WML functions

Many WML functions accept an optional argument named **class**. In particular, the HTML markup functions and HTML character markup functions accept the **class** argument. The class argument is a keyword argument; its syntax is: the argument's name, a colon, and the value of the argument.

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>"
```

button(*string1, url, string2,* class:*classname*)

Parameters *string1* and *string2* are character strings, and *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 the first parameter as anchor text pointing to the address specified by *url*. If *url* is **""**, then *string2* is used as anchor text. If *string2* is not specified, and *url* is **""**, then no button is returned.

For example:

```
filename(up) --> "doc_9.html"
button("[Up]",filename(up)) -->
   "<A HREF="doc_9.html" class:classname>[Up]</A>"

filename(next) --> ""
button("[Next]",filename(next),"[Next]") --> "[Next]"
```

See also **link(string,url, class:classname)**.

center(*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 within the **CENTER** HTML markup.

For example:

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

comment(*string*)

Parameters *string* is a character string.

Returns the HTML markup of string as an HTML comment.

For example:

```
comment("toto") --> "<!-- toto -->"
```

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).

For example:

```
glossdescription("Programming Techniques")
--> "<DD CLASS=classname>Programming Techniques"
```

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

heading(*level, string,* class:*classname*)

Parameters *level* is an integer between 1 and 6. *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 heading of *level*.

For example:

```
heading(1,"Chapter 1") --> "<H1 CLASS=classname>Chapter 1</H1>"
```

hrule(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 horizontal rule.

For example:

```
hrule() --> "<HR CLASS=classname>"
```

image(*url, keyword-args*)

Parameters *url* is a URL, which can also be a pathname of an image file. The keyword arguments are all optional, and may be given in any order. For information about the meaning of these keywords arguments, see the HTML specification. The keyword arguments include:

align:*alignment (*where *alignment* is **top**, **bottom**, **middle**, **left**, or **right**)

width:*integer*

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:

```
link("CERN","http://www.cern.ch/") -->
        "<A HREF="http://www.cern.ch/" CLASS=classname>CERN</A>"
```

See also **button(string1, url, string2, class:classname)**.

listitem(*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 for *string* as a list item (within a list environment).

For example:

```
listitem("item 1") --> "<LI CLASS=classname>item 1"
```

paragraph(*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 for *string* as a simple paragraph.

For example:

```
paragraph("Some text") --> "<P CLASS=classname>Some text</P>"
```

span(*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 within the **SPAN** HTML markup.

For example:

```
span("toto", class:doggy) --> "<span class:doggy>toto</span>"
```

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>"
```

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

For example:

```
italic("toto") -> "<I CLASS=classname>toto</I>"
```

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

For example:

```
keyboard("toto") -> "<KBD CLASS=classname>toto</KBD>"
```

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

For example:

```
sample("toto") -> "<SAMP CLASS=classname>toto</SAMP>"
```

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

For example:

```
small("toto") -> "<SMALL CLASS=classname>toto</SMALL>"
```

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

For example:

```
strong("toto") -> "<STRONG CLASS=classname>toto</STRONG>"
```

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

For example:

```
subscript("toto") -> "<SUB CLASS=classname>toto</SUB>"
```

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

For example:

```
strikethrough("toto") -> "<STRIKE CLASS=classname>toto</STRIKE>"
```

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.

parameters:*string* where *string* contains all the parameters to the Java applet. Usually, this string is constructed by using **concatenate** on a set of calls to **appletparameter**.

Returns the HTML code that supports a Java applet. For example:

```
@NavApplet=applet
  (appletcode: "WMNavigator.class",
   width: 500, height: 135,
   parameters:
     concatenate(
       appletparameter("next", filename(next)),
       appletparameter("previous",filename(previous)),
       appletparameter("up", filename(up)),
       appletparameter("top",filename(top)),
       appletparameter("toc",filename(TOC)),
       appletparameter("index",filename(INDEX)),
       appletparameter("parseabletoc", filename(TOC)),
       appletparameter("indentstring", "??")
                ),
   default: @NavPanel
  );
```

appletparameter(*name, value***)**

Parameters *name* is a string that names the parameter to the Java applet. *value* is the value of that parameter.

Returns the HTML code for a parameter to a Java applet.

In the following example, we assume that the function call **filename(next)** returns **"guide2.html"**:

```
appletparameter("next", filename(next))
--> <PARAM NAME="next", VALUE="guide2.html">
```

Index

F

G

H

I

O

P

Q

R

S

T

U

V

W