MML Reference

Frame Technology Corporation
333 West San Carlos Street
San Jose, California 95110
USA

Frame Technology International Limited
3 Furzeground Way
Stockley Park
Uxbridge
Middlesex UB11 1DE
United Kingdom

May 1995

FRAME

Important Notice

MML Reference Contents

Go To

To get help on using this manual, click here.

To go to a section, click on a topic below.

Print Manual

1 Introduction

Go To

Frame® publishing software from Frame Technology® Corporation includes support for a markup language called MML (Maker Markup Language). You can use any standard text editor to create an MML file. Later, you can open the MML file as a Frame document or import it into a Frame template. In a department where different people are responsible for writing and formatting documents, writers can use MML statements to mark up manuals in progress; at the same time, graphic designers can create the formatting specifications in Frame templates.

MML supports many formatting and layout features of Frame documents. For example, you can use MML to specify:

- Document page size, document margins, and number of columns (including the Custom Blank Paper options to the New command)
- Header and footer layouts
- A Paragraph Catalog (including most options available in the Paragraph Designer)
- Font definitions (including most options available in the Character Designer)
- Document text with varying character and paragraph formats (all document text in an MML file is assumed to comprise one text flow)
- Anchored frames containing graphics
- Markers (including all options available in the Marker window)

MML cannot define the following (you can add them after you open or import the MML file):

- Unanchored frames and graphics
- Imported text
- Irregular column layouts
- Column layouts that vary from page to page
- Multiple text flows
- Multiple-line or multiple-font headers and footers
- Multiple master pages
- Tables
- Equations
- Side heads
- Stored character formats
- Paragraph, footnote, table, or anchored frames that straddle two or more columns
- Conditional text

- Variables
- Color
- Structure elements
- Text runarounds
- Dashed line patterns

Using this manual

This manual contains:

- General instructions for creating and using MML files
- A complete description of each MML statement
- Sample MML files
- MML error messages

FrameMaker® and FrameMaker+SGML™ both read MML files. In this manual, the term *Frame document* refers to a document created by either FrameMaker or FrameMaker+SGML. This manual contains information for the UNIX, Macintosh, Windows, and NeXT versions of FrameMaker and FrameMaker+SGML.

MML files

An MML file is a standard ASCII text file containing MML statements and document text. You can create the file with any standard text editor.

After you import or open an MML file, you can modify, print, and save it using Frame document commands. If you open the MML file or import it by copying, any changes you make in the Frame document are not reflected in the original MML file. Thus, if you want the MML file to serve as the master source for the document, you must make the changes to the MML file.

If you import the MML file by reference into a Frame document, you can continue to use the original MML file as the master source for the document. Each time you open the Frame document, it interprets the MML file and updates the resulting text.

If you use your Frame product to create or edit an MML file, save the file as Text Only using the Save As command. To open an MML file as a text file, hold down a modifier key and click Open in the Open dialog box.

In this version	Use this modifier key
UNIX	Shift
Macintosh	Option
Windows	Control
NeXT	Alt

In UNIX versions of a Frame product, an MML filename must end with the file suffix `.mml` (`.framemml` on NeXT computers). This suffix alerts your Frame product that the file is an MML file and needs to be interpreted before it is imported into, or opened as, a Frame document.

UNIX versions of a Frame product use the `mmltomif` program to interpret MML files. You can also run `mmltomif` directly to interpret MML files. The `mmltomif` program accepts optional command-line arguments. It has the following syntax:

`mmltomif -L`*`language`* `-I`*`include_path input_file output_file`*

`language`	language in use, such as `usenglish`
`include_path`	pathname for included files (you can specify multiple include paths by specifying `-I`*`include_path`* for each path you want to search)
`input_file`	pathname of MML file to read
`output_file`	pathname of MIF file to write (if you specify this option, you must also specify the *`input_file`* option)

Macintosh and Windows versions do not require an `.mml` suffix; the `<MML>` statement identifies a file as an MML file, and a dynamic filter processes the file.

Using MML to create Frame documents

You can use MML to:

- Specify the content of a document for which formatting information is stored in a Frame template.

  You use a small subset of MML instructions to specify when to use paragraph formats and when to change the character format for words and phrases. You create the paragraph formats and set up the document layout in a Frame template.

- Specify both the content and format of a document.

  You use more complex MML statements to define formatting and layout specifications in the MML file.

When you use MML to create Frame documents, use two MML files to describe a document: an MML *include file* contains formatting information, and an MML *document content file* contains document text. Using two files makes it easier to correct errors. In addition, you can use one include file to create several documents with the same formatting.

If you use a Frame template to specify formatting, your include file can be very brief. It lists the paragraph formats in the template's Paragraph Catalog and any character formats and MML macros you want to use.

If you want to keep formatting information and document content in one file, the file should contain the information that would appear in an include file followed by the information that would appear in a document content file.

Specifying document format with a Frame template

The easiest way to use MML is to specify formatting information in a Frame template. In addition to the template, you use a simplified include file and a document content file to specify the document text.

For a complete description of the sections in an MML file, see "MML file structure" on page 7. For a sample Frame document created from a template, an include file, and a document content file, see "Specifying document format with a template" on page 22.

Setting up the template

Open a Frame template, set up the document layout, and create paragraph formats. For information about creating templates, see your user's manual.

Creating the include file

Use a standard text editor to create the include file. It should contain:

- An MML identification line
- A Macro Definition section
- A `<!DefineTag>` statement for each paragraph tag in the template (see "Paragraph statements" on page 12)
- Font definitions for character format changes to be used for words or phrases in the document content file

Creating the MML document content file

Use a standard text editor to create the document content file. It should contain:

- An MML identification line (required)
- An `<Include>` statement that names the MML include file (see "Control and macro statements" on page 9)
- A Document Text section

Importing the MML file into the template

To create a Frame document from the MML file, use the New command to create a new document from a template. Then use the Import>File command to import the MML document content file into the document. Use the Save As command to save the resulting document under a new filename.

You can also open the MML document content file as a Frame document and use the Import>Formats command from file menu to copy formats from the template into the new document. If you use this method, turn on the option to remove format overrides (see your user's manual).

Specifying document format with MML

When you use an include file and a document content file to create a Frame document, the include file describes document formatting; the document content file contains the document text.

For a complete description of the sections in an MML file, see "MML file structure" on page 7. For a sample document created from include and document content files, see "Specifying document format with MML" on page 25.

Creating an include file

Use a standard text editor to create an include file describing document formatting. It should contain the following sections:

- An MML identification line
- A Macro Definition section
- A Font Definition section
- A Paragraph Format Definition section
- A Document Layout section

Creating a document content file

Use a text editor to create the document content file. It should contain:

- An MML identification line (required)
- An `<Include>` statement that names the MML include file (see "Control and macro statements" on page 9)
- A Document Text section

Opening the document content file

To create a Frame document from the include and document content files, open the document content file with your Frame product. When you save the resulting document, a Frame product replaces the `.mml` file suffix with a `.doc` file suffix to avoid overwriting the original MML file.

2 MML Statements

Go To

An MML file consists of markup statements and document text. Markup statements begin with a left angle bracket (<) and end with a balancing right angle bracket (>). For example, <Section> signals the beginning of a new section, while <Family Times> switches fonts. Case is not significant in statement names, so <FaMiLy Times> would work as well.

All text outside angle brackets is document text. Within document text, adjacent nonblank lines are considered to be in the same paragraph; one or more blank lines separate paragraphs (a blank line is two consecutive return characters). <Paragraph> markup statements can also signal paragraph boundaries. (See "Paragraph statements" on page 12.)

If the text contains a left or right angle bracket character (that is, one that should appear in the Frame document instead of beginning or ending a markup statement), a backslash character must precede the angle bracket (for example, \< or \>).

MML file structure

An MML file can contain the following sections, in this order.

Section	Contains
MML identification line	An <MML> statement identifying the file as an MML file.
Macro Definition	<!DefineMacro> and <!DefineChar> statements that define simple macros used in subsequent markup statements.
Font Definition	<!DefineFont> statements that define named sets of font properties for use within document text and other markup statements.
Paragraph Format Definition	<!DefinePar> and <!DefineTag> statements that define or declare paragraph formats.
Document Layout	Document layout statements that define document properties.
Document Text	ASCII characters along with font, special character, anchored frame, and paragraph-related markup statements. The first nonspace text character or document text statement outside a markup statement signals the beginning of the document text section.

All sections except the MML identification line and the Document Text section are optional. For information about which sections to include in an MML file, see "Using MML to create Frame documents" on page 3.

Markup statement overview

The general format of a markup statement is:

<StatementName OptionalDataItems>

The following conventions are used to describe the format of data items.

This term	Means
char	A single character code or a backslash equivalent, such as `\t` for tab (see "MML character set," next).
string	Any sequence of characters enclosed by double quotation marks (`"abc"`). To use a backslash in a string, type `\\`. To use double quotation marks in a string, type `\"`.
commentstring	Any sequence of characters. Double quotation marks are not required.
name	A simple alphanumeric string starting with a nonspace character and ending with a space or a right angle bracket (>). Case is not significant.
number	An integer.
boolean	`Yes`, `No`, `Y`, or `N`. Case is not significant.
measure	A real number, which may contain a decimal point and digits to the right of the decimal point, followed by an optional unit of measurement: `inch`, `in`, `"` (for inch), `cm`, `mm`, `pica`, `pc`, or `pt`. If a unit of measurement is not provided, the MML filter uses the default unit (see `<Unit>` on page 10).
unit	A unit of measurement: `inch`, `in`, `"` (for inch), `cm`, `mm`, `pica`, `pc`, or `pt`.
lrcd	Alignment type: `left`, `l`, `right`, `r`, `center`, `c`, `leftright`, `lr`, `decimal`, or `d`.

MML character set

MML uses the Frame character set. However, because an MML file can contain only ASCII characters and because of MML parsing requirements, you must represent certain characters in document text with backslash (\) sequences.

Character	Representation
Tab	`\t`
Forced Return	`\n`
<	`\<`
>	`\>`
\	`\\`
non-ASCII	`\x`*nn*

The backslash sequence \xnn represents a Frame character code, a 1-digit or 2-digit hexadecimal number that represents a character outside of the printing ASCII range. The backslash sequence must end with a space. You can use character codes in the ranges \x20 to \x7e and \x80 to \xfe. Other values are ignored. For an explanation of character code values, see the user's manual or the *Quick Reference* for your Frame product.

Control and macro statements

You use control and macro statements to set up your MML file. The following statements (except for the <MML> statement) can appear anywhere in the file.

<MML *commentstring*>
All MML files must begin with an <MML> statement.

<Comment *commentstring*>
Places comments in the MML file. The *commentstring* argument is ignored when the file is read.

<!DefineMacro *name string*>
Replaces all occurrences of the newly defined statement <*name*> with the replacement text in *string*. Note that *string* is rescanned each time *name* is encountered.

> **Important:** You must put one or more spaces between the *name* and *string* arguments when you define a macro.

For example, suppose your text editor does not give you a way to type the Yen symbol (¥). You can use the following statement to define a new MML statement, <Yen>, that represents the Frame Yen character (\xb4):

```
<!DefineMacro Yen "<Character \\xb4 >">
```

When you import or open your document in a Frame product, MML replaces each <Yen> statement with a Yen character.

For a list of character codes, see the user's manual or the quick reference guide for your Frame product.

> **Important:** MML statement names are not case sensitive, so you should not use macro names that are case sensitive.

For example, suppose you define two macros for the symbols É and é as follows:

```
<!DefineMacro E@' "<Character \\x83 >">
<!DefineMacro e@' "<Character \\x8e >">
```

When you open this file in a Frame product, MML replaces all <E@'> and <e@'> statements with the symbol é. Because statement names are not case sensitive, MML reads both <!DefineMacro> statements as definitions of the same macro, and it uses the most recently defined macro.

<!DefineChar *char string*>
Replaces all occurrences of *char* in the document text with *string*.

> **Important:** You must put exactly one space between !DefineChar and *char* and at least one space between *char* and *string*.

Use this statement to remap character codes for foreign and other special keyboards. For example, suppose that the Yen symbol is represented by character code \xfe in the MML file, but the Yen symbol is represented by character code \xb4 in a Frame document. This statement:

```
<!DefineChar \xfe "<Character \\xb4 >" >
```

causes MML to convert all \xfe characters in the MML file to \xb4 characters in the Frame document. (See <Character> on page 18.)

<Include *string*>
Reads the file named *string* as MML input. If you provide just a filename, MML searches for the file in the directory containing the MML file being processed. If you provide a complete pathname, MML searches for the file in the directory specified by the pathname.

<Units *unit*>
Establishes default units for all measurements. If a <Units> statement appears in the MML file, it must come before the font definitions section. If the statement is not supplied, the default value inch is used for all measurements.

<!Alias *newname currentname*>
Creates a new statement name that is a synonym for an existing statement name. The *newname* and *currentname* arguments are not enclosed in angle brackets. For example, you could define a synonym and then use the synonym to define a macro as follows:

```
<!Alias !MD !DefineMacro>
<!MD bi "<Bold><Italic>">
```

You could then use <bi> within the document text to set the current font to bold italic.

<EndOfInput>
Ignores all remaining text in the MML file. Use <EndOfInput> to debug an MML file or to temporarily modify an MML file so that a Frame product reads only part of it.

<Message *string*>
Prints the specified *string*. Use <Message> to debug an MML file.

In the UNIX versions of Frame products, messages appear in the window from which you started your Frame product. In the Windows and Macintosh versions, you must turn on Show File Translation Errors in the Preferences dialog box to display messages. The messages appear in a console window in Microsoft Windows and in an Error Log window on the Macintosh.

Font statements

MML font statements provide character format control similar to the control provided by the Character Designer window or by the Format command in a Frame document. You cannot use MML statements to store character formats in the Character Catalog.

Most of the font statements can appear in the Font Definition, Paragraph Format Definition, Document Layout, and Document Text sections of an MML file. However, `<!DefineFont>` can appear only in the Font Definition section.

<family *name*>
Changes font family. The *name* argument must match a font family name that is installed with a Frame product; case is significant. The font families are listed in the Format>Font submenu. If no `<family>` statement is provided, the default family is `Times`.

<italic>
<noitalic>
<bold>
<nobold>
<underline>
<nounderline>
<strike>
<nostrike>
<oblique>
<nooblique>
These statements turn various font styles on and off. A font style remains in effect until you turn it off. For example, the following MML input:

```
You can switch from <bold> bold to <nobold> plain font styles.
```

produces this result in a Frame document:

You can switch from **bold to** plain font styles.

The `<italic>` and `<oblique>` statements are synonymous, as are the `<noitalic>` and `<nooblique>` statements.

<plain>
Same as `<nobold>` `<noitalic>` `<nounderline>` `<nostrike>`. The default style is `<plain>`.

<superscript>
<subscript>
<normal>
These statements change the relative position of the text baseline. The baseline position remains in effect until you turn it off. The default position is `<normal>`.

For example, the following MML input:

```
e<superscript>i*pi<normal>=-1
```

produces this result in a Frame document:

e^{i*pi}=-1

<pts *number*>
Changes font size. For example, <pts 10> changes the current font size to 10 points. The default size is <pts 12>.

<!DefineFont *name fontstatements*>
Defines a character format. It executes the list of *fontstatements* (the statements defined in this section) and then establishes the current font properties as the character format. For examples of <!DefineFont> and its use, see Appendix A, "Samples."

The character formats you define in an MML file are used to indicate font changes for words and phrases. However, they do not correspond to the formats stored in a document's Character Catalog, so you cannot store formats in the Character Catalog or apply a character format by using the tags of formats stored in the Character Catalog.

Paragraph statements

Paragraph statements in MML provide a subset of the paragraph formatting control provided by the Paragraph Designer.

Most paragraph statements can appear within the Paragraph Format Definition section and between paragraphs in the Document Text section. Exceptions are <!DefinePar> and <!DefineTag>, which can appear only in the Paragraph Format Definition section.

<par>
Ends a paragraph. The current font properties and paragraph settings remain in effect. Two or more consecutive return characters act as a <par> statement. A new paragraph begins only when a nonspace text character is read; leading tabs and spaces are ignored. To begin a paragraph with a tab, use the predefined <Tab> macro (see <Character> on page 18). To begin a paragraph with a space, use the predefined <HardSpace> macro or define your own <Space> macro by using the <!DefineMacro> statement (see <!DefineMacro> on page 9). The <par> statement is most useful within macro definitions.

<LeftIndent *measure*>
Changes the paragraph left indent. The default value is 0".

<RightIndent *measure*>
Changes the paragraph right indent. The default value is 0".

<FirstIndent *measure*>
Sets the left indent for the first line of a paragraph. The default value is `0"`.

<SpaceBefore *measure*>
Sets the space above the paragraph. The default value is `0pt`.

<SpaceAfter *measure*>
Sets the space below the paragraph. The default value is `0pt`.

<Leading *measure*>
Determines the space between lines within the paragraph. The default value is `2pt`.

> **Important:** Always specify a unit in a *measure* argument. If you don't, MML uses the current default unit, which is set by the `<Units>` statement and is usually inches. So `<Leading 2>` would put 2 inches of leading between each paragraph, not 2 points.

<Alignment *lrc*>
Sets the alignment of paragraph lines. The default value is `lr` (justified).

<AutoNumber *boolean*>
Sets automatic numbering of paragraphs. If `<AutoNumber Yes>` is specified, there must also be a valid `<NumberFormat>` string. The default value is `No` (no automatic numbering).

<NumberFormat *string*>
Determines the numbering format for paragraphs that are automatically numbered. Ignored unless `<AutoNumber Yes>` is specified. The default value is `""` (an empty string).

A paragraph autonumber format in an MML file can contain a number series label, printing characters, and counters. However, MML supports a limited form of autonumbering. To use the full functionality of autonumbering in a Frame document, set up the autonumber format in a Frame template rather than in an MML file (see your user's manual).

To specify an autonumber format in the MML file, specify an optional series label, a counter, and printing characters. Use the following syntax for counters.

Use	To
A plus sign (+)	Increase the value of the counter by 1
A number sign (#)	Use the current value of the counter
A number	Set the value of the counter to the specified number

For example, the following table shows three autonumber formats for section, subsection, and sub-subsection headings.

This autonumber format	Appears in the document as	
`<NumberFormat "Section +.0\t">`	Section 1.0	Section heading
`<NumberFormat "Section +.0\t">`	Section 2.0	Next section heading
`<NumberFormat "Section #.+\t">`	Section 2.1	Subsection heading

This autonumber format	Appears in the document as	
`<NumberFormat "Section #.+\t">`	Section 2.2	Next subsection heading
`<NumberFormat "Section #.#.+\t">`	Section 2.2.1	Sub-subsection heading
`<NumberFormat "Section #.#.+\t">`	Section 2.2.2	Next sub-subsection heading
`<NumberFormat "Section +.0\t">`	Section 3.0	Next section heading

<Hyphenate *boolean*>
Turns automatic hyphenation on or off. The default value is `Yes` (automatic hyphenation).

<ColumnTop *boolean*>
Sets the starting place for the paragraph on the page. If `<ColumnTop Yes>` is specified, the paragraph starts at the top of a column; otherwise it starts anywhere. These settings correspond to the Start Anywhere and Start Top of Column properties in the Paragraph Designer. The default value is `No` (paragraph starts anywhere).

<WithNext *boolean*>
Determines whether or not to keep the paragraph in the same column as the beginning of the next paragraph. The default value is `No` (don't keep paragraphs together).

<Tolerance *number*>
Specifies the maximum number of adjacent lines that may be hyphenated when automatic hyphenation is turned on. The default value is `2`.

<BlockSize *number*>
Specifies the minimum number of widow and orphan lines. The default value is `1`.

<TabStop *dimension*>
Sets a tab stop at the indicated position. This statement can only appear in a `<TabStops>` statement.

<TabStopType *lrcd*>
Establishes the tab type for all subsequently defined tab stops until the next `<TabStopType>` statement. This statement can only appear in a `<TabStops>` statement. The default value is `l` (left tab).

<TabStopLeader *char*>
Establishes the tab leader character for all subsequently defined tab stops until the next `<TabStopLeader>` statement. This statement can only appear in a `<TabStops>` statement. The default value is the space character (no leader).

> **Important:** You must have exactly one space between `TabStopLeader` and the *`char`* argument when you define a tab stop leader.

<TabStops *tabstatements*>
Defines a set of tab stops. Each tab stop is determined by a <TabStop> substatement. The tab type and associated leader character are determined by the most recent <TabStopType> and <TabStopLeader> substatements, which may be freely intermingled among the <TabStop> substatements, as shown in the following example:

```
<TabStops
 <TabStopType l>
 <TabStopLeader .>
 <TabStop .5">
>
```

To clear all tabs, use an empty tab stop list: <TabStops>.

<!DefinePar *name parstatements*>
Creates a named paragraph format that has the paragraph properties specified in the list of *parstatements* (the statements defined in this section, "Paragraph statements").

When you open or import an MML document, the resulting Frame document contains a Paragraph Catalog entry for each paragraph format defined in the MML file using <!DefinePar> statements. A stored paragraph format is applied to any MML paragraph that is preceded by the format's tag. For examples of the <!DefinePar> statement and its use, see "Include file" on page 25.

<!DefineTag *name*>
Establishes a paragraph format name like <!DefinePar>. However, unlike <!DefinePar>, <!DefineTag> does not generate a Paragraph Catalog entry when the MML file is imported or opened as a Frame document.

Use <!DefineTag> when you want to import an MML file into a Frame document that already has the Paragraph Catalog set up or when you will import formats from a template. When an MML paragraph is preceded by a tag declared by a <!DefineTag> statement, the Frame document's Paragraph Catalog is searched for a format with a matching tag. If such a format exists, the paragraph's format is set to match the corresponding format in the Paragraph Catalog. For examples of the <!DefineTag> statement and its use, see "Include file" on page 22.

> **Important:** MML tag names cannot have a space in them (although tag names in a Frame document can). The *name* argument must match a tag name in the Paragraph Catalog; case is significant.

Document layout statements

MML's document layout statements provide control similar to the control provided by the Custom Blank Paper options to the New command and by the Column Layout and Page Size commands. (See your user's manual.)

Document layout statements may appear only in the document layout section.

<PageWidth *measure*>
Sets the page width. The default value is `8.5"`.

<PageHeight *measure*>
Sets the page height. The default value is `11"`.

<TopMargin *measure*>
<BottomMargin *measure*>
<LeftMargin *measure*>
<RightMargin *measure*>
Sets the page's top, bottom, left, and right margins. Each margin is offset from the corresponding edge of the paper and defines the area occupied by a text frame. The default value for each margin is `1"`.

<Columns *number*>
Sets the number of columns. The default value is `1`.

<ColumnGap *measure*>
Determines the gap between columns. The default value is `0.25"`.

<LeftHeader *string*>
<CenterHeader *string*>
<RightHeader *string*>
<LeftFooter *string*>
<CenterFooter *string*>
<RightFooter *string*>
Establishes the specified *`string`* as part of a page header or page footer (left-aligned, centered, or right-aligned). The default value is `""` (an empty string).

To insert a page number variable in a header or footer, use a number sign (#) in the string.

<HeaderFont *fontstatements*>
Designates the specified font statements or a named font definition to be used in all header and footer strings. The default value is `<Family Times> <pts 12> <Plain>`.

<HeaderTopMargin *measure*>
Specifies the margin from the top edge of the paper to the header. The header sits just below the margin. The default value is `0.5"`.

<HeaderBottomMargin *measure*>
Specifies the margin from the bottom edge of the paper to the baseline of the footer. The default value is `0.5"`.

<HeaderLeftMargin *measure*>
Specifies the margin from the left edge of the paper to the header and footer. The default value is `1"`.

<HeaderRightMargin *measure*>
Specifies the margin from the right edge of the paper to the header and footer. The default value is `1"`.

<HeaderPageNumberStyle *style*>
Specifies the document's main page numbering style where *`style`* is `Arabic`, `UCRoman`, `LCRoman`, `UCAlpha`, or `LCAlpha`. The default value is `Arabic`.

<FirstPageHeader *boolean*>
Controls whether or not headers are displayed on the first page of a document. The default value is `Yes` (display headers).

<FirstPageFooter *boolean*>
Controls whether or not footers are displayed on the first page of a document. The default value is `Yes` (display footers).

<DoubleSided *boolean*>
Specifies single-sided or double-sided pagination. `No` means single-sided. The default value is `No`.

<FirstPageLeft *boolean*>
Specifies a left or right first page. `No` means the first page is considered a right page. `<FirstPageLeft>` is meaningful only if preceded by a `<DoubleSided Yes>` statement. The default value is `No` (first page is a right page).

<FirstPageNumber *number*>
Sets the number for the first page of the document. The default value is `1`.

Document text statements

The Document Text section contains:

- Text outside of markup statements
- Font statements and references to named character formats (see `<!DefineFont>` on page 12)
- Paragraph statements and references to named paragraph formats (see `<!DefinePar>` on page 15 and `<!DefineTag>` on page 15)
- Macros defined with `<!DefineMacro>` and `<!DefineChar>` (see "Control and macro statements" on page 9)
- `<Character>` statements (described in this section)
- Anchored frames defined with `<AFrame>` statements (described in this section)
- Markers defined with `<Marker>` statements (described in this section)

Regular document text in an MML file can only contain ASCII characters. To include special characters in regular document text, use a backslash sequence (see "MML character set" on page 8) or use the `<Character>` statement (described next).

<Character *number*>
Represents a character code value in the ranges 32 to 126 and 128 to 254 (`\x20` to `\x7e` and `\x80` to `\xfe`). Other values are ignored. To use hexadecimal values in a `<Character>` statement, leave a space between the number and the right bracket (for example, `<Character \x86 >`). Use `<Character>` statements to enter characters outside the printing ASCII range. They may occur within document text and within definitions of macros that are used in document text. Whenever `<Character>` statements are nested within `<!DefineMacro>` and `<!DefineChar>` statements, you must type two backslashes before the hexadecimal value. For example:

```
<!DefineChar \xfe "<Character \\xb4 >" >
```

Two backslashes are necessary because of the order in which the MML filter processes the statement. Note that `!DefineChar` must be followed by exactly one space.

You can also use the following predefined macros, which expand to the appropriate `<Character>` statements.

Macro name	Description
`<Tab>`	Tab
`<HardSpace>`	Nonbreaking space
`<DiscHyphen>`	Discretionary hyphen
`<NoHyphen>`	Suppress hyphenation
`<Cent>`	Cent (¢)
`<Pound>`	Sterling (£)
`<Yen>`	Yen (¥)
`<EnDash>`	En dash (–)
`<EmDash>`	Em dash (—)
`<Dagger>`	Dagger (†)
`<DoubleDagger>`	Double dagger (‡)
`<Bullet>`	Bullet (•)
`<HardReturn>`	Forced return

<AFrame <BRect *l t w h*>>
Creates an anchored frame, placing the anchor symbol after the character that precedes the `<AFrame>` statement. The `<AFrame>` statement must contain a `<BRect>` statement that gives the frame's left and top coordinates relative to the enclosing page or frame and the frame's width and height. Following the `<BRect>` statement, there may be other substatements, including the `<FrameType>` statement (used to define the frame's position

relative to the anchor symbol). The <AFrame> statement, and all its substatements, are MIF (Maker Interchange Format) statements. For information about MIF statements, see the online manual *MIF Reference.*

A minimal <AFrame> statement is:

```
<AFrame <BRect 0 0 4" 2"> >
```

This statement places an empty 4-inch by 2-inch anchored frame in the document. The default frame type is <FrameType Below>, which corresponds to the Below Current Line setting in the Anchored Frame dialog box. (See your user's manual.)

For an example of an <AFrame> statement that includes graphics, see "Document content file" on page 28.

<Marker *MarkerSubstatements*>
Inserts a marker. You can use the following MIF <Marker> substatements to describe the marker's settings; other MIF <Marker> substatements are not allowed.

Statement	Marker setting	
<MType *number*>	Specifies the marker type number. The marker type numbers correspond to the marker names in the Marker window as follows:	
	0	Header/Footer $1
	1	Header/Footer $2
	2	Index
	3	Comment
	4	Subject
	5	Author
	6	Glossary
	7	Equation
	8	Hypertext
	9	Cross-Ref
	10	Conditional Text
	11 through 25	Type 11 through Type 25
<MText *string*>	Specifies the marker text. The string must begin with a left single quotation mark (`) and end with a straight single quotation mark ('). You cannot use double quotation marks (").	

For example:

```
<Marker <MType 2> <MText `rectangles, drawing'> >
```

describes an index entry with the text "rectangles, drawing."

To include a left angle bracket (<), right angle bracket (>), or backslash (\) in the marker text, precede it with a backslash. For example:

```
<Marker <MType 2>
  <MText `duplicate, see copy\<$nopage\>'> >
```

describes an index marker whose marker text is “duplicate, see copy<$nopage>”.

For more information about MIF `<Marker>` substatements, see the online manual *MIF Reference*.

Obsolete statement

The following MML statement is obsolete:

`<ForceFont>`

MML reads a `<ForceFont>` statement but ignores it.

A Samples

Go To

This appendix contains sample MML descriptions of documents. The sample documents have been provided with this manual and are located in the directory or folder in which your Frame product is installed.

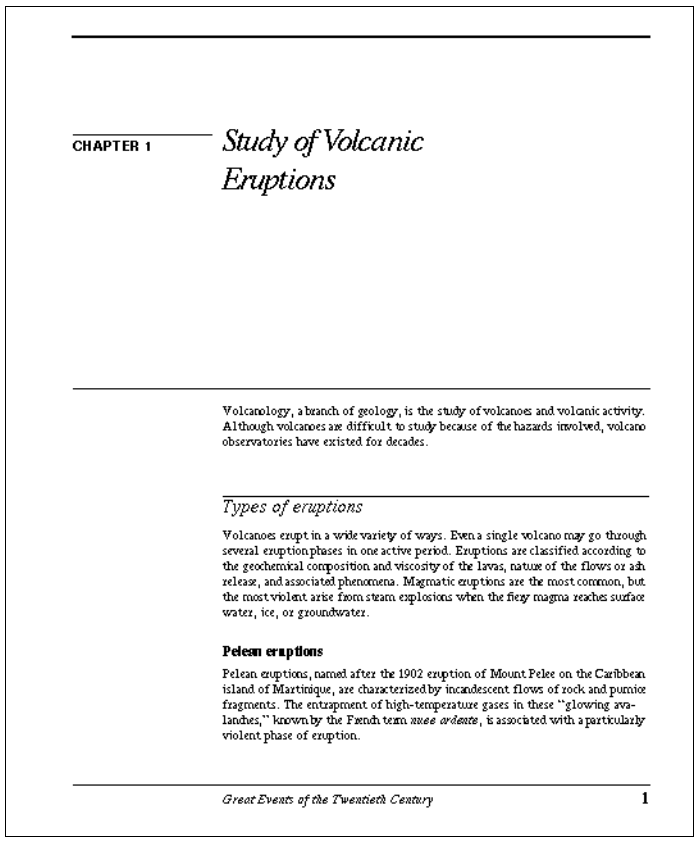
For this Frame product version	Look here
UNIX	The `/fminit/`*`language`*`/Samples` directory, where *`language`* is the language in use, such as `usenglish`
Macintosh	The `Samples` folder
Windows	The `samples` directory

The MML sample files have the file suffix `.mml`. On the NeXT computer, the files have the suffix `.framemml`.

Specifying document format with a template

The following illustration shows a document created by importing an MML file into the standard Frame template `Book/Chapter` (`book/chapter` in Windows).

To create this sample document, use the New command to create a new document from the Book/Chapter template. Then use the Import>File command to import the MML sample file `chaptxt.mml` into the document.



CHAPTER 1

Study of Volcanic Eruptions

Volcanology, a branch of geology, is the study of volcanoes and volcanic activity. Although volcanoes are difficult to study because of the hazards involved, volcano observatories have existed for decades.

Types of eruptions

Volcanoes erupt in a wide variety of ways. Even a single volcano may go through several eruption phases in one active period. Eruptions are classified according to the geochemical composition and viscosity of the lavas, nature of the flows or ash release, and associated phenomena. Magmatic eruptions are the most common, but the most violent arise from steam explosions when the fiery magma reaches surface water, ice, or groundwater.

Pelean eruptions

Pelean eruptions, named after the 1902 eruption of Mount Pelee on the Caribbean island of Martinique, are characterized by incandescent flows of rock and pumice fragments. The entrapment of high-temperature gases in these "glowing avalanches," known by the French term *nuee ardente*, is associated with a particularly violent phase of eruption.

Great Events of the Twentieth Century 1

Include file

The following include file, `chapfmt.mml`, contains `<!DefineTag>` statements for paragraph formats in the Chapter template.

```
<MML>
<Comment *** Include file for Chapter template>
<Comment *** Define paragraph formats in the Book/Chapter template>
<!DefineTag Body>
<!DefineTag BodyAfterHead>
<!DefineTag Bulleted>
<!DefineTag BulletedCont>
```

```
<!DefineTag CellBody>
<!DefineTag CellHeading>
<!DefineTag ChapterTitle>
<!DefineTag Equation>
<!DefineTag Extract>
<!DefineTag Figure>
<!DefineTag Footnote>
<!DefineTag Heading1>
<!DefineTag Heading2>
<!DefineTag HeadingRunin>
<!DefineTag Numbered>
<!DefineTag Numbered1>
<!DefineTag NumberedCont>
<!DefineTag TableFootnote>
<!DefineTag TableTitle>
```

Document content file

The following file, `chaptxt.mml`, contains the chapter text; each paragraph is tagged with one of the formats defined in the include file.

```
<MML>
<Comment *** Include formats in chapfmt.mml>
<Comment *** On the NeXT computer, it's chapfmt.framemml>
<Include "chapfmt.mml">

<Comment *** Report Body>
<ChapterTitle>
<Marker <MType 2> <MText `volcanic eruptions\<$startrange\>'> >Study
of Volcanic Eruptions

<BodyAfterHead>
Volcanology, a branch of geology, is the study of volcanoes and
volcanic activity. Although volcanoes are difficult to study because
of the hazards involved, volcano observatories have existed for
decades.

<Heading1>
Types of eruptions

<BodyAfterHead>
Volcanoes erupt in a wide variety of ways. Even a single volcano may
go through several eruption phases in one active period. Eruptions
are classified according to the geochemical composition and
viscosity of the lavas, nature of the flows or ash release, and
associated phenomena. Magmatic eruptions are the most common, but
the most violent arise from steam explosions when the fiery magma
reaches surface water, ice, or groundwater.

<Heading2>
<Marker <MType 2> <MText `eruptions, Pelean'> >Pelean eruptions

<BodyAfterHead>
Pelean eruptions, named after the 1902 eruption of Mount Pelee on
the Caribbean island of Martinique, are characterized by
incandescent flows of rock and pumice fragments. The entrapment of
high-temperature gases in these ``glowing avalanches,'' known by
the French term <italic>nuee ardente<noitalic>, is associated with
a particularly violent phase of eruption. <Marker <MType 2> <MText
`volcanic eruptions\<$endrange\>'> >
```

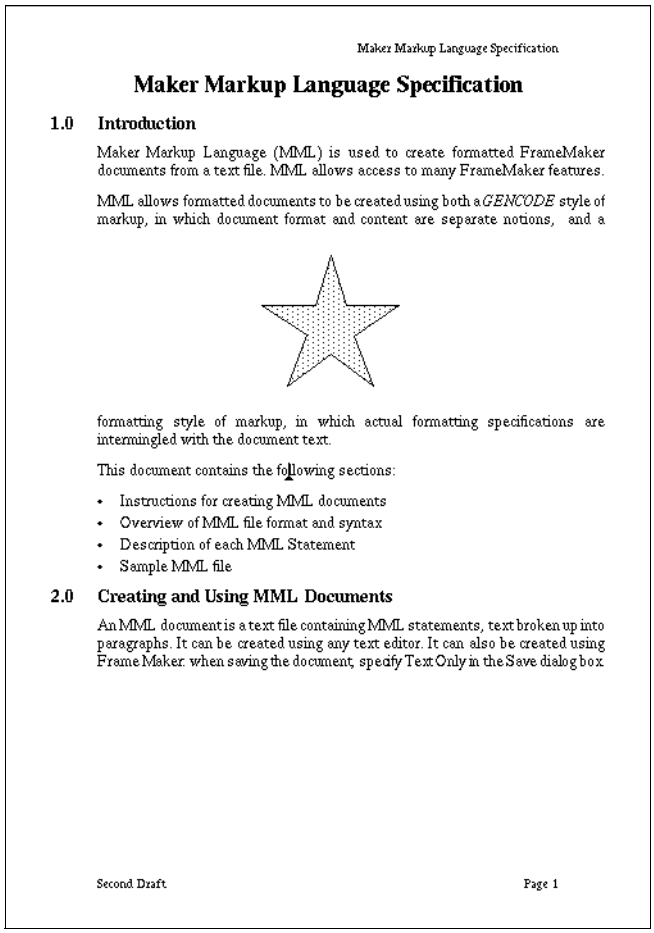
Specifying document format with MML

The following illustration shows a document created from two MML files:

- An include file contains document formats.
- A document content file contains the document text.

The two files can be merged; however, you should keep format information and document content in separate files.

To create this sample document, open the MML file `sample.mml`.


Maker Markup Language Specification

Maker Markup Language Specification

1.0 Introduction

Maker Markup Language (MML) is used to create formatted FrameMaker documents from a text file. MML allows access to many FrameMaker features.

MML allows formatted documents to be created using both a *GENCODE* style of markup, in which document format and content are separate notions, and a

formatting style of markup, in which actual formatting specifications are intermingled with the document text.

This document contains the following sections:

- Instructions for creating MML documents
- Overview of MML file format and syntax
- Description of each MML Statement
- Sample MML file

2.0 Creating and Using MML Documents

An MML document is a text file containing MML statements, text broken up into paragraphs. It can be created using any text editor. It can also be created using FrameMaker: when saving the document, specify Text Only in the Save dialog box.

Second Draft Page 1

Include file

The following include file, `formats.mml`, contains formatting information.

```
<MML 1.00 - Sample standard font, paragraph, and document formats>
<Comment *** Define fonts for Title, Section, Body, Headers
   and Footers.  Most of the defaults are good, so we just
```

```
   specify family, size, and style. "ft" stands for "font
   for Titles", "fs" is "font for sections," etc.>
<!DefineFont ft
  <Family Times>
  <pts 18>
  <Bold>
>
<!DefineFont fs
  <Family Times>
  <pts 14>
  <Bold>
>
<!DefineFont fb
  <Family Times>
  <pts 12>
  <Plain>
>
<!DefineFont fhf
  <Family Times>
  <pts 10>
  <Plain >
>
<Comment *** Set appropriate font for a Title paragraph and
   define its format.>
<!DefinePar Title
 <ft>
 <Alignment Center>
 <SpaceAfter 12pt>
>
<Comment *** Set font and define other paragraph formats.>
<!DefinePar Section
  <fs>
  <Alignment Left >
  <LeftIndent 0.50">
  <FirstIndent 0.00">
  <RightIndent 0.00">
  <Leading 0pt>
  <SpaceBefore 9pt>
  <SpaceAfter 9pt>
  <AutoNumber Yes >
  <NumberFormat "+.0\t">
  <TabStops <TabStop 0.50"> >
>
```

```
<!DefinePar Body
  <fb>
  <Alignment LeftRight >
  <LeftIndent 0.50">
  <FirstIndent 0.50">
  <RightIndent 0.00">
  <Leading 2pt>
  <SpaceBefore 0pt>
  <SpaceAfter 10pt>
  <AutoNumber No >
  <TabStops>
>
<!DefinePar BulletItem
  <Alignment Left >
  <LeftIndent 0.75">
  <FirstIndent 0.50">
  <RightIndent 0.00">
  <Leading 2pt>
  <SpaceBefore 0pt>
  <SpaceAfter 3pt>
  <AutoNumber Yes>
  <NumberFormat "\xA5 \t">
  <Comment *** \xA5 is the bullet character. \t is a tab
     character.>
  <TabStops <TabStop 0.75">>
>
<Comment *** Document Layout descriptions. Most of the
   default settings are good. >
<Pagewidth 7.00">
<PageHeight 10.00">
<TopMargin 0.75">
<BottomMargin 0.75">
<LeftMargin 0.50">
<RightMargin 0.50">
<HeaderTopMargin 0.40">
<HeaderBottomMargin 0.46">
<HeaderLeftMargin 1.00">
<HeaderRightMargin 1.00">
```

Document content file

The following MML file, `sample.mml`, contains the document text.

```
<MML 1.00 A sample mml file>

<Comment *** Include the font, paragraph, document definitions
   from another file.  By keeping the formats in different files
   than the document text, all documents can be assigned a new
   format by just changing one file.>

<Comment *** Include formats in formats.mml>
<Comment *** On the NeXT computer, it's formats.framemml>

<Include "formats.mml">

<Comment *** Define a few macros just to show how it is done.
   Would normally put such standard macros in an include file.>
<!DefineMacro if "<Italic>">
<!DefineMacro pf "<Plain>" >
<!DefineMacro bf "<Bold>"  >

<Comment *** Set up Headers and Footers. The next line sets the
   font.>
<HeaderFont <fhf>>
<RightHeader "Maker Markup Language Specification">
<LeftFooter "Second Draft">
<RightFooter "Page #">

<Comment *** Start of Document Text ***>
<Title>
Maker Markup Language Specification
<Section>
Introduction
<Body>
Maker Markup Language (MML) is used to create formatted
Frame documents from a text file. MML allows access to
many Frame product features.

<Comment *** The following Body paragraph contains an anchored
   frame.  The AFrame statement is equivalent to a MIF AFrame
   statement. (For a detailed description, see the online
   manual "MIF Reference.")  Inside the frame is a star. We
   just show this here so you can see how it is done.>

MML allows formatted documents to be created using both a
```

```
<if>GENCODE <pf>style of markup, in which document format and
content are separate notions,
<AFrame <BRect 0 0 4 2> <FrameType Below>
  <Polygon
   <Pen 0> <PenWidth `1.0'> <Fill 6> <Inverted No >
   <NumPoints 10>
   <Point  2.03" 0.29"> <Point  2.19" 0.83"> <Point  2.76" 0.83">
   <Point  2.28" 1.17"> <Point  2.49" 1.71"> <Point  2.03" 1.36">
   <Point  1.56" 1.71"> <Point  1.76" 1.15"> <Point  1.28" 0.83">
   <Point  1.86" 0.83">
  > # end of Polygon
>
and a formatting style of markup, in which actual formatting
specifications are intermingled with the document text.

This document contains the following sections:
<BulletItem>
Instructions for creating MML documents

Overview of MML file format and syntax

Description of each MML Statement

Sample MML file

<Section>
Creating and Using MML Documents
<Body>
An MML document is a text file containing MML statements, text
broken up into paragraphs. It can be created using any text
editor.  It can also be created using a Frame product: when saving
the document, specify Text Only in the Save dialog box.

<Comment *** Would be followed by additional such lines>
```

B MML Messages

Go To

Frame products use an MML filter to read an MML file. The filter translates the MML file and produces a temporary MIF file that a Frame product opens as a document. While the filter is reading the MML file, it might detect errors such as unexpected character sequences. It responds by displaying error messages. Even if it finds an error, the filter continues to process the MML file and reads as much of the document as possible.

This section lists the messages produced by the filter, along with their explanations. Words in italic indicate variable words in a message. A line number is printed along with most messages when they appear on the screen.

***MML* MSG: *Message_String*.**
Not an error; generated by a user `<Message>` statement.

***MML*: Bad option *'Character'*.**
The filter did not recognize this option character. The option is ignored.

Bad Boolean: *'Unexpected_String'*.
The filter expected to see `Yes` or `No`. The value `No` is assumed.

Bad lrcd: *'Unexpected_String'*.
The filter expected to see `Left`, `Right`, `Center`, `Decimal`, or `LeftRight`.

Bad real number: *'Unexpected_Char'*.
A nonreal number character appeared in the middle of a real number.

Bad style: *'Unexpected_String'*.
The filter expected to see `Arabic`, `LCRoman`, `UCRoman`, `LCAlpha`, or `UCAlpha`.

Bad unit: *'Unexpected_String'*.
The filter expected to see a valid unit (`inch`, `cm`, and so on).

Cannot find *'filename'*.
The filter can't find the specified input file. Make sure that the file exists, and that you have read access to it; then try again.

Cannot find end of comment on line *n*.
The comment that began on the specified line did not end by the time the file was completely read.

Cannot open *filename*.
The filter couldn't find the named include file. Make sure that the file is in the correct format and that you have read access to it; then try again. If this message still appears, close some open files or windows and try again.

Cannot open temporary file.
The filter couldn't open its temporary file. Make sure you have write access to `/tmp`, your home directory, or the current directory; then try again. If this message still appears, close some open files or windows and try again.

Cannot write *filename*.
The filter couldn't open the specified output file for writing. Make sure you have write access; then try again. If this message still appears, close some open files or windows and try again.

Character *'Character'* needs ending space.
Characters specified with `\x` must end with a space.

Expected string, not *'Unexpected_Char'*.
The filter expected to see a string starting with a double quotation mark (") rather than the unexpected character shown in the message.

FATAL!
The filter encountered a problem from which it can't recover. Write down the error message and contact Frame Technical Support.

Input stack overflow.
There are too many nested include files (maybe in an include loop). The maximum nesting depth is 100.

Internal Error.
The filter encountered an internal error. Please contact Frame Technical Support.

Junk at end of command: *Junk_String*.
The filter expected to see a a right angle bracket (>).

Keyword too long: over 1000 characters.
While looking for a macro name or other keyword, the filter found a very long token (over 1,000 characters). Check the MML file for a syntax error and try again.

Never finished defining *'Character'*.
The filter encountered a `<!Define…>` statement within a `<!Define…>` statement (for example, a `<!DefineChar>` statement within a `<!DefinePar>` statement). You must finish the first `<!Define…>` statement before beginning another.

The filter ignores the first `<!Define…>` statement and continues reading the file. The results, however, are not likely to be what you intended.

Out of memory!
The filter was unable to complete the translation of the MML file because it ran out of memory. To free memory for the filter, quit some Frame document windows or terminate other processes.

String too long.
A very long string was found. The maximum string length is 1000 characters; characters beyond that are truncated.

Tab commands allowed only within <TabStops . . .>.
The statements `<TabStopType>`, `<TabStopLeader>`, and `<TabStop>` can appear only within a `<TabStops>` statement.

Too many -I options.
The maximum number of `-I` options is 100. The `mmltomif` filter exits. (This error message is valid only for UNIX versions of Frame products.)

Undefined macro: *Macro_Name*.
There is no definition for this macro. The undefined macro is ignored.

Unexpected right angle bracket.
A right angle bracket (>) with no matching left bracket (<) has appeared.

Usage: mmltomif [-L language] [-I path] [input [output]].
You started `mmltomif` in a shell window with incorrect parameters. Restart `mmltomif` with the following parameters:

language	language in use, such as `usenglish`
path	pathname for included files (you can specify multiple include paths by specifying `-I`*path* for each path you want to search)
input	pathname of MML file to read
output	pathname of MIF file to write (if you specify this option, you must also specify the *input* option)

This error message is valid only for UNIX versions of Frame products.

MML Reference Index

Go To

To go to a page, click on a page number below.