WATCOM Script/GML

Tutorial and Reference Manual

D.S. McKee
J.W. Welch

WATCOM International Corporation
Waterloo, Ontario, Canada

Disclaimer

WATCOM (WATCOM International Corp. and all of its subsidiaries) makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall WATCOM, its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands or claim for lost profits, fees or expenses of any nature or kind.

Printed in Canada

WATCOM	Telephone:	(519) 886-3700
415 Phillip Street	FAX:	(519) 747-4971
Waterloo, Ontario	BBS:	(519) 884-2103
CANADA N2L 3X2		

Preface

This manual describes the text preparation language GML (Generalized Markup Language). The language was originally developed by IBM for use with the DCF (Document Composition Facility) product distributed by IBM. The language is used in many installations as a convenient means of specifying the components of the input text to be formatted into documents.

WATCOM Script/GML may be used for all stages of document preparation. WATCOM Script/GML may also be used as a "debugging" document processor during the draft stages of a document. The direct processing of GML tags provides a precise diagnosis of errors and faster processing of the document. Waterloo Script or IBM DCF can then be used to produce the final copy of the document.

This document was prepared using WATCOM Script/GML on an 486 PC with the Apple LaserWriter printer.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

List of Figures

List of Figures

List of Figures

List of Figures

Tutorial

1 Introduction to GML

1.1 What is GML?

GML (**G**eneralized **M**arkup **L**anguage) is a language by which the components of a document are specified. A GML user creates a computer file containing a document specification, and then causes a GML processor (a computer program) to be executed. As the GML processor executes, the specification file is read and the document is produced for an output device such as the terminal or a printer.

Only in rare situations is the document completed at this point. Usually, there are revisions which must be made. The user enters these revisions by modifying the original specification file and then causes the GML processor to be executed to obtain a revised version of the document. This revision process is typically repeated many times.

The method outlined above where an input file containing document content information is used to produce a document is often called **text formatting**. GML and many text formatting languages use this method. There are a number of benefits to using the GML Language:

1. A GML document is described using high-level entities such as headings, paragraphs and lists. On the other hand, most text formatting languages have formatting commands which describe how a document is formatted. This latter approach does not enforce consistency and often restricts a document to one particular style.

2. GML is used to describe the components of a document. These component descriptions do not contain specifications about the appearance of the components once they have been formatted. Thus, the specification file is essentially independent of the layout used. A **layout** is a set of rules to determine the way in which a document is to be formatted. Examples of these rules are the number of lines to be displayed on a page, and the style and placement of page numbers on each page. A default layout is supplied with the GML processor; optionally, a GML user may create additional layouts for situations in which the default is not satisfactory. A common method of document preparation is to use one layout to create a completed version of a document and then to use a specialized layout in the final production run of the document.

3. The GML language has been found to be easy to learn. A user of the language seldom becomes involved in details concerning layout specifications since there often exists a layout suitable for use with a given document.

The file(s) containing the GML specifications for a document are usually prepared and revised using a text editor. This manual does not describe the use of a specific editor.

4 What is GML?

2 Getting Started

The GML tutorial shows you how to use the GML language with WATCOM Script/GML. You are encouraged to try the examples as they are introduced. Trying variations of the examples will also help in understanding the GML language.

The tutorial does not describe every feature of the GML language. The intent is to "get you going"; once some familiarity with the language has been achieved, the reference section may be used to obtain complete details about other parts of the language.

Do not be concerned with the style of the document while you are learning the GML language. One of the benefits of GML is the ability to change the style of a document after it has been entered. The document style is defined by the layout feature in WATCOM Script/GML, and is described in a later chapter.

Since any editor capable of creating and modifying text files may be used to prepare the GML input, this manual does not describe the use of a specific editor. It is recommended that you become familiar with the use of the editor before attempting the examples in this chapter.

2.1 First Example Document

The first example illustrates a number of the features in the GML language. The GML file is as follows:

```
:GDOC.
:BODY.
:H0.Simple Document
:P.This is the first sentence of the
very first paragraph.
This is the second sentence in
that paragraph.
:P.Here is the second paragraph.
This is the second
sentence in the second paragraph.
:eGDOC.
```

Figure 1. *First GML example*

The document, when processed, may appear as follows:

```
Simple Document

   This  is the  first  sentence  of the
very  first   paragraph.   This  is  the
second sentence in that paragraph.

   Here is the second paragraph.    This
is the  second  sentence  in the  second
paragraph.
```

***Figure 2.** : Output of* Figure 1

The first example consists of a heading and two paragraphs.

It should be noted that the GML source contains two types of information.

Tags Each tag in the example starts with a colon(:) and ends with a period(.). Although the period is usually optional, it is a good practice to always include it. This convention avoids the necessity of learning the specific instances in which the period would be required. The tags used in the first example are **:gdoc**, **:body**, **:h0**, **:p** and **:egdoc**. Tags can be entered using either upper or lower case letters. The tags will be shown in upper case in the tutorial examples. Ending tags, such as :eGDOC., will be entered with the "e" in lower case to emphasize that they are used in conjunction with other tags.

Text The words and punctuation to be processed is called text.

The following tags have been introduced in this example.

:gdoc This tag indicates the start of the GML document. It must precede all text to be formatted.

:egdoc This tag indicates the end of the GML document. It must be the last tag in the GML source file.

:body This tag indicates the start of the main text for the document. Since this is a simple document, it precedes the specification of the document text.

6 First Example Document

:h0 This tag specifies a level zero heading. Heading tags define the structural divisions of the document text, and are described in the section "Headings" on page 44.

:p This tag indicates the start of a paragraph. The sentences of the paragraph may start on separate input lines; WATCOM Script/GML processes the text into a paragraph for you. It is a common practice to start sentences on separate input lines to make it easier to modify the text in future revisions of the document.

With most of the examples in the tutorial, the layout of the resulting output shown in this book will be slightly different from the tutorial output you will see. This difference results from the use of a modified version of the default layout for the examples in this book.

2.2 Processing the Examples

The simplest way to process the examples given in this tutorial is to use one of the command formats shown below. The format of the command will depend on the system on which you are running the WATCOM Script/GML processor. For a complete description of the WGML command and the command line options, see "Running WATCOM Script/GML" on page 207.

Create the file `EX1.GML` with an editor, using the document text shown in Figure 1 on page 5. Note that the source for these examples is supplied with the product distribution.

The **WGML** command is used to invoke the WATCOM Script/GML processor. The command line option **DEVICE** selects the type of device for which the document is being processed.

IBM VM/CMS	WGML EX1 (DEVICE TERM
DEC VAX/VMS	WGML EX1/DEVICE=TERM
IBM PC/DOS	WGML EX1 (DEVICE TERM

If you see an error message concerning invalid options or device/font member not found, there has probably been an error in the installation of WATCOM Script/GML. Refer to the section of the installation guide on setting up the device library for corrective action.

It is worthwhile attempting to create a simple document at this point. The document should consist of a number of paragraphs. Include a sufficient amount of text so that more than one page of output is generated. WATCOM Script/GML will automatically

format the document into pages. With some experimentation, it may also be noted that WATCOM Script/GML will not create a page in which only the first line of a paragraph occurs at the bottom of the page. When this situation arises, the entire paragraph is placed on the succeeding page. This is called **widowing** (widows are lonely items which appear all by themselves; WATCOM Script/GML attempts to prevent widows as it formats documents).

8 Processing the Examples

3 WATCOM Script/GML Screen Displays

This chapter describes the screens presented by WATCOM Script/GML before and during the processing of a document. These screen displays are presented when working on a IBM PC/DOS or compatible computer system. You may omit reading this chapter if you are using WATCOM Script/GML on an IBM VM/CMS or DEC VAX/VMS computer system.

The WATCOM Script/GML document processing system is composed of two programs. The WGML program produces a formatted document for a specified output device. The screens displayed by this program are discussed in the first section. The WGMLUI (**W**ATCOM GML **U**ser **I**nterface) program provides a mechanism for working with your source document and selecting the options for the WGML program. The screens displayed by WGMLUI are discussed in the second section.

You may choose to start either WGMLUI or the WGML program. It is suggested that you use WGMLUI to process your documents until you have become familiar with the WGML options.

3.1 Processing Displays

A status screen is presented by WATCOM Script/GML while it is processing a document. Accumulated statistics and information about the document section currently being processed is displayed.

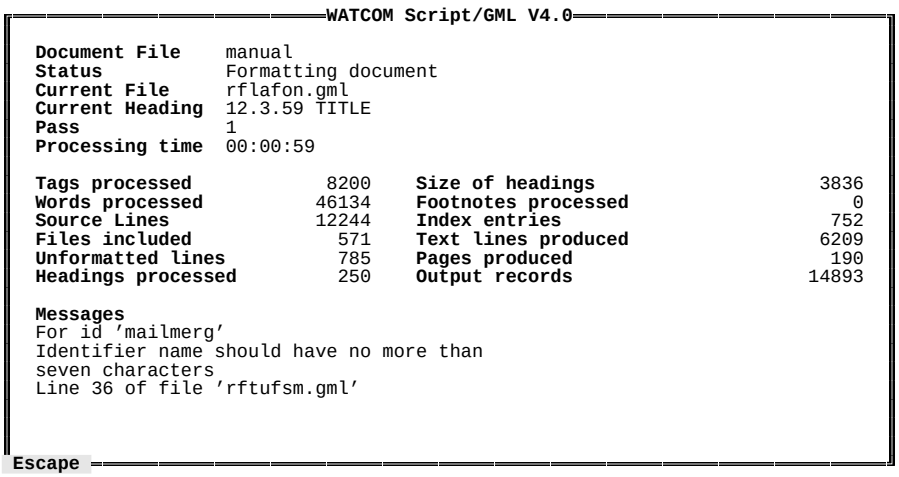

Figure 3. *WATCOM Script/GML Status Screen*

To stop document processing before WGML has completed formatting:

Mouse:

Move the mouse pointer to the **Escape** area at the bottom left corner of the screen and click. An error screen indicating that processing has been stopped will be displayed. Point to the **Enter** area at the bottom right corner of the screen and click.

Keyboard:

Press the **Escape** key. Alternatively, pressing the **Ctrl** and **Break** keys at the same time will stop document processing. After pressing the key(s), an error screen will be displayed. Press the **Enter** key to continue.

NOTE: WATCOM Script/GML can be used with or without a mouse pointer device. Throughout this chapter both methods are illustrated using the style shown above. If you are using a mouse, read the paragraph marked "Mouse:", otherwise, read the paragraph marked "Keyboard:".

3.1.1 Display Areas

The top third of the screen displays the status of the document being processed. Each information area is updated to indicate the part of the document WATCOM Script/GML is processing.

The middle third of the screen displays document statistics. When processing is completed, this area will show the statistics for the entire document. The message 'Text lines produced' reflects the number of lines in the formatted document which are not blank. The message 'Output records' reflects the number of records sent to the output device. With some devices, many formatted document lines may be sent to the device in one output record.

The bottom third of the screen displays WATCOM Script/GML messages. These messages are warnings about possible errors in the source document.

3.1.2 Error Screen

When an error is detected in the GML document, an error screen of the following form is displayed:

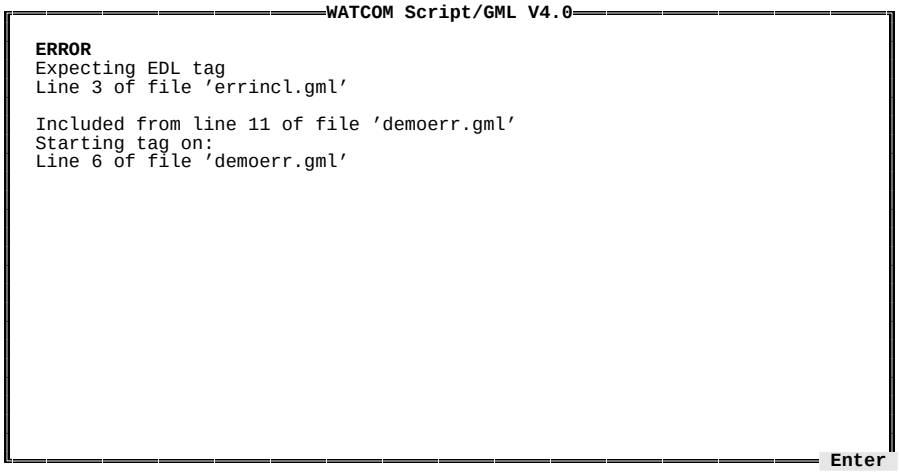

Figure 4. *WATCOM Script/GML Error Screen*

In this example the ending tag of a definition list is missing. WATCOM Script/GML shows where in the document source the error occurs and at what point in the document the starting tag (:DL in this case) was specified. After reading the message:

Mouse:

Move the mouse pointer to the **Enter** area at the bottom right corner of the screen and click.

Keyboard:

Press the **Enter** key.

3.2 WATCOM Script/GML User Interface Displays

The main document screen is displayed when WGMLUI is first started. There is one input area where you may enter the name of the GML source document. In the following document screen, the file name `manual` has been entered.

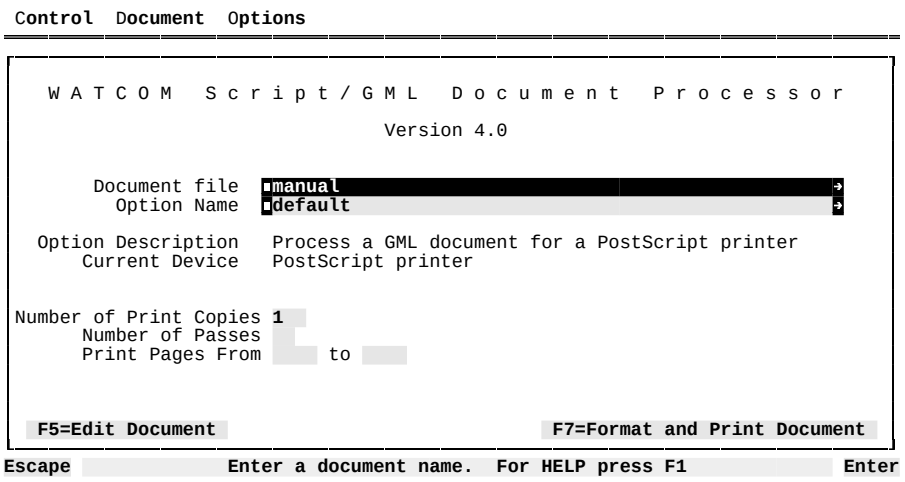

Figure 5. *Main Document Screen*

At this point you can process the document with the selected options.

Mouse:

Point to the highlighted area near the bottom of the screen labeled 'F7=Format and Print Document' and click.

Keyboard:

Press the **F7** key.

NOTE: The highlighted areas with labels such as 'F7=Format and Print Document' are called *hotspots*. The actions indicated by the hotpots can be activated by either selecting on the hotspot with a mouse pointer, or by pressing the key indicated by the label. Further references to hotspots in this document will be of the form 'select the *Format and Print Document* hotspot'.

The formatted document is directed to a specific output location. This may be a physical device such as the printer or terminal screen, or a file on the disk. The device selected for this tutorial is the PostScript printer, which directs the output to a file with the same name as the source document and a `PS` file type. In the previous example, the disk file `manual.ps` would contain the formatted output.

Pressing the **F7** key causes the document to be formatted. If there are no errors, the resulting document file is printed. If the printing process seems to take a very long time, the most common reason is that a printer is not connected to your computer. In most cases, an error will be reported after approximately ninety seconds.

3.2.1 Obtaining Help

While you are learning WGMLUI, the most useful feature of the system will probably be the help facility. Pressing the **F1** key will give you help on the area of the screen which is currently selected. Pressing the **F1** key again will tell you how to use the help facility. After pressing the **F1** key twice you will see the following screen:

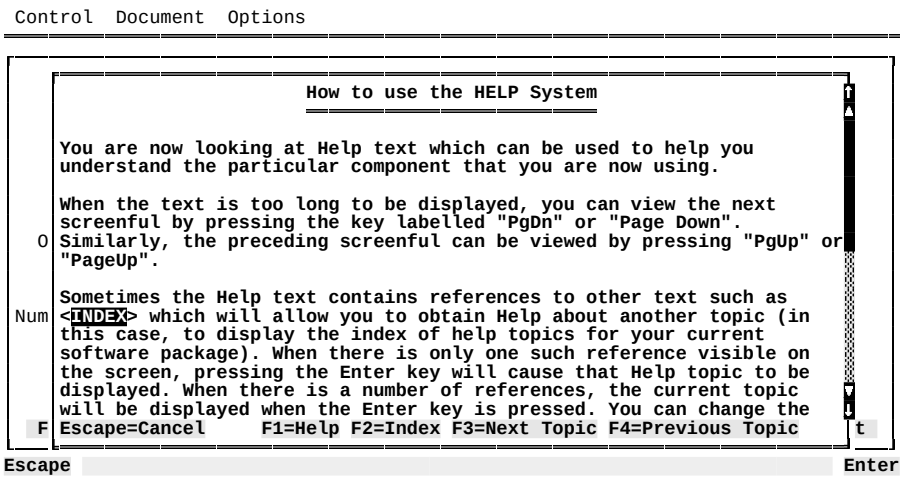

Figure 6. *Help about the Help Facility*

Press the **Escape** key to remove the help screen. Note that the bottom line on the screen contains an area displaying a hint on how to proceed. The hint line is updated as you select different areas on the screen.

Help on the current area may also be obtained by selecting *Help* from the *Control* menu. The menu bar appears on the top line of the screen. Each menu name can be 'pulled down' to select from a number of menu item choices.

Mouse:

Point to the *Control* menu and press the mouse button. Drag the mouse to the *Help* item and release the button. Pressing the **Escape** key will return you to the document screen.

Keyboard:

While pressing the **Alt** key, pressing the **C** key will pull down the *Control* menu. Use the cursor keys to select the *Help* item and press **Enter**. Pressing the **Escape** key will return you to the document screen.

While pressing the **Alt** key, pressing the highlighted letter of a menu name will pull down that menu.

3.2.2 Control Menu

All of the screens displayed by WGMLUI have *Control* as the first menu on the menu bar. The *Help* menu item was described in the previous section.

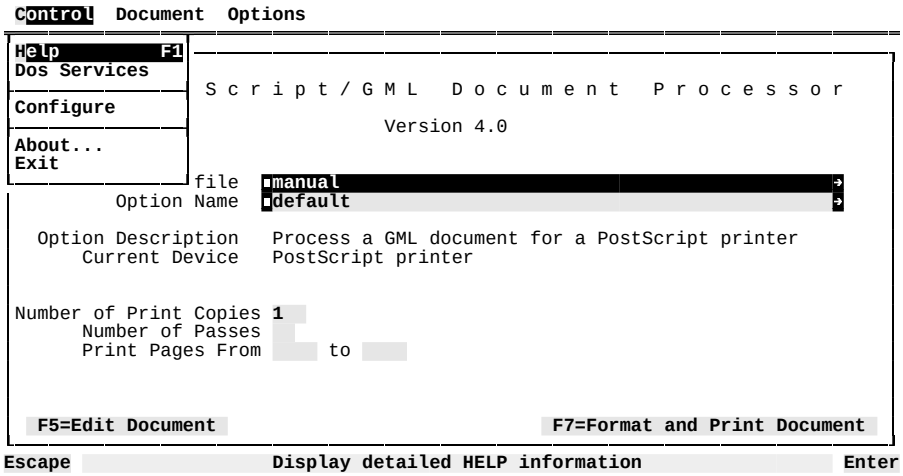

Figure 7. *Control menu*

Selecting the *DOS services* menu item will place you in the DOS environment. Enter the `exit` command to return from DOS. The *Configure* menu item is selected when you wish to change some of the default values used by the WGMLUI program, and is discussed in more detail in a later section (see "Changing the Configuration" on page 27). The *About...* menu item displays some information about the WGMLUI program. Selecting *Quit* is the same as pressing the **Escape** key, except while in the main document screen. Selecting *Quit* in the main screen will exit the WGMLUI program.

3.2.3 Selecting/Editing a Document

Selecting the *Select a different document* menu item from the *Document* menu will display on the screen a file browser for GML source files.

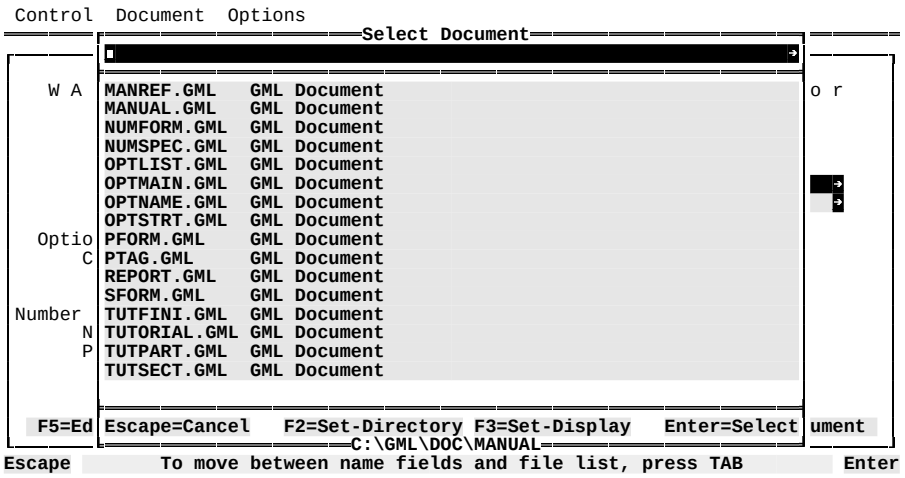

Figure 8. *Select Document file browser*

The file browser presents you with an area at the top of the browser where you may enter a new name. The available files are shown in a list, with the file extension of 'gml' being used to selectively display the files in the directory. If your files have an extension other than 'gml', you may edit the file pattern to specify a different pattern (such as *.doc) by pressing the **F3** key. You may also choose a new file from the list displayed on the screen.

Mouse:

Move the mouse pointer to the appropriate file name and click. If there are more source files than can be displayed at one time, click on the bar along the right side of the browser screen to view the other files.

Keyboard:

Use the **Tab** and cursor keys to select the appropriate file name. The file name area at the top of the browser will be updated to show the currently selected file. The cursor down (↓) and cursor up (↑) keys may be used to view files beyond the browser window.

After entering or selecting the appropriate file name, press **Enter** to place the name in the document file name area.

Selecting the *Edit Document* hotspot will also bring up the file browser. The name area of the browser will contain the name of the last edit file (or the main document name if

you are using the edit function for the first time). Pressing **Enter** after selecting a file name will edit the file. If you are using the WATCOM Editor, entering `exit` will save the file and return you to WATCOM Script/GML.

A number of different select operations will use the file browser to obtain a file name. Selecting a file is performed in the same way for each case.

3.2.4 Printing a Document

An output file is usually created as the document is processed. By selecting the *Print a document file* menu item from the *Control* menu, the document may be reprinted at a later time.

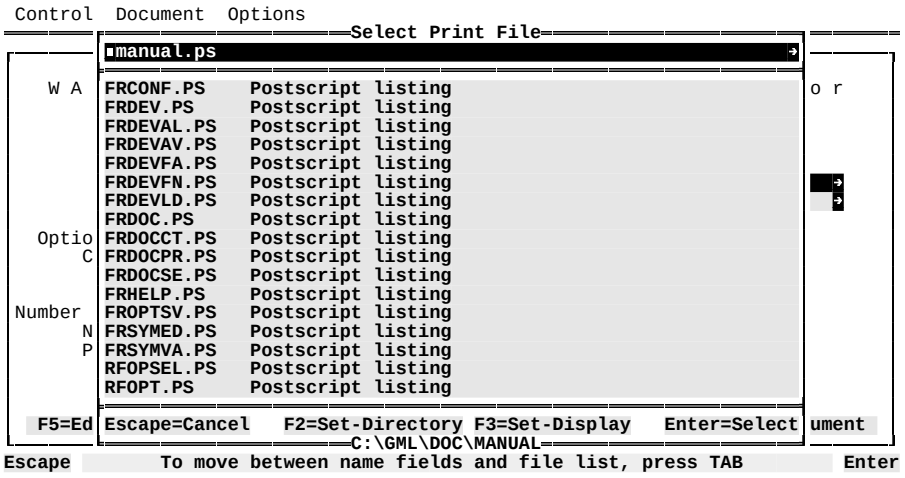

Figure 9. *Print a Document*

The name area of the browser will contain the name of the output file for the currently selected document. Your may enter or select from the browser file list another name for printing. Pressing the **Enter** key will copy the file to the printer.

NOTE: Some network systems require a separate command to actually send the file to the printer. You must issue this command yourself by selecting *DOS Services* from the *Control* menu. This will place you in DOS where you can issue the command. Entering the `exit` command will return you to WATCOM Script/GML. See "Changing the Configuration" on page 27 for information on configuring the print action.

3.2.5 Changing the Options

Selecting the *Edit current options* menu item from the *Options* menu will display the option screen. Options determine how the WGML processor will format the document, select different methods for displaying processing information, select device and font information, and define symbols before processing begins. See "Running WATCOM Script/GML" on page 207 for a detailed description of the options and their usage.

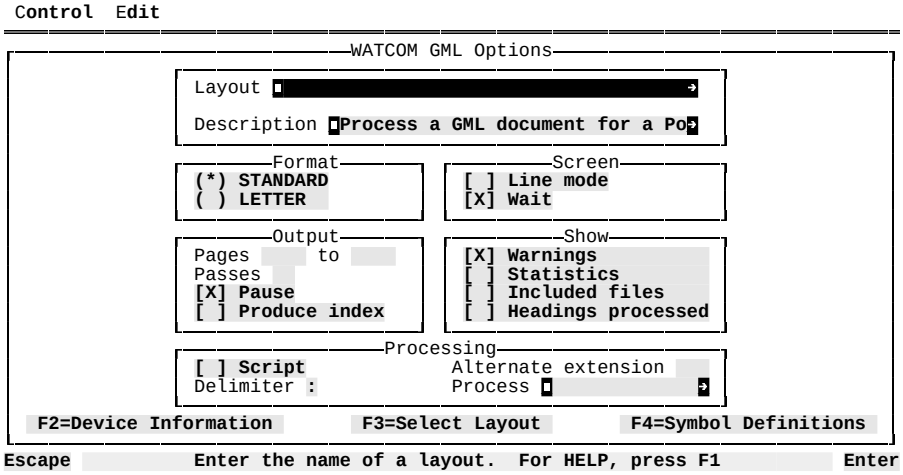

Figure 10. *Option Screen*

The current area on the screen is for entering the name of a layout file. Pressing **F3** will invoke the file browser for selecting the layout name. Some of the other areas on the screen are *check boxes* and *radio buttons*. Check boxes have an indicator at the left side to show if the option is ON or OFF. Radio buttons are similar to check boxes, but appear in groups. Only one of the buttons may be ON at any one time.

Mouse:

Move the mouse pointer to the *LETTER* button in the *Format* group and click.

Keyboard:

Press the **Tab** key twice. You will be positioned in the *Format* group at the *LETTER* button. Press the ↓ key.

This will select letter tag processing for your document.

Mouse:

Move the mouse pointer to the *Line mode* check box in the *Screen* group and click.

Keyboard:

Press the **Tab** key. You will be positioned in the *Screen* group at the *Line mode* check box. Press the space bar.

This will select the line mode for displaying information. Each message from the WGML processor is displayed individually on the standard DOS screen.

After these changes, your screen should appear as follows:

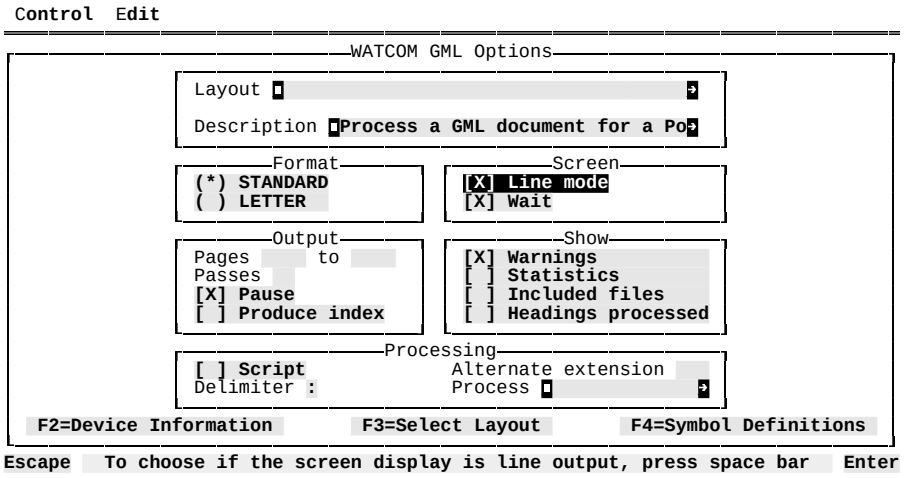

Figure 11. *Option Changes*

Pressing the **Enter** key will accept the changes and return you to the main document screen. Pressing the **Escape** key will return you to the document screen without the changes.

3.2.6 Selecting Device Information

WATCOM Script/GML must produce the output for a specific device when a document is formatted. Associated with a device is a default output file and a set of fonts. The default output file is either a device (such as lpt1), a full file name, or a file pattern (such as *.ps) which specifies how to create the output file name from the document name. The set of fonts define the character sets and their attributes used to produce the

text in the document. The fonts numbered zero through three are used by the GML tags :HP0 through :HP3. Font numbers greater than three may be referenced in the layout or by the :SF GML tag.

NOTE: The device and font lists shown in the following examples may not be the same as the lists shown on your screen. As part of the WATCOM Script/GML product installation, the devices and fonts available at your site are selected. Although the PostScript fonts are used in the following examples to illustrate font selection, the operations performed are the same for any device.

From the option screen, select the *Device Information* hotspot.

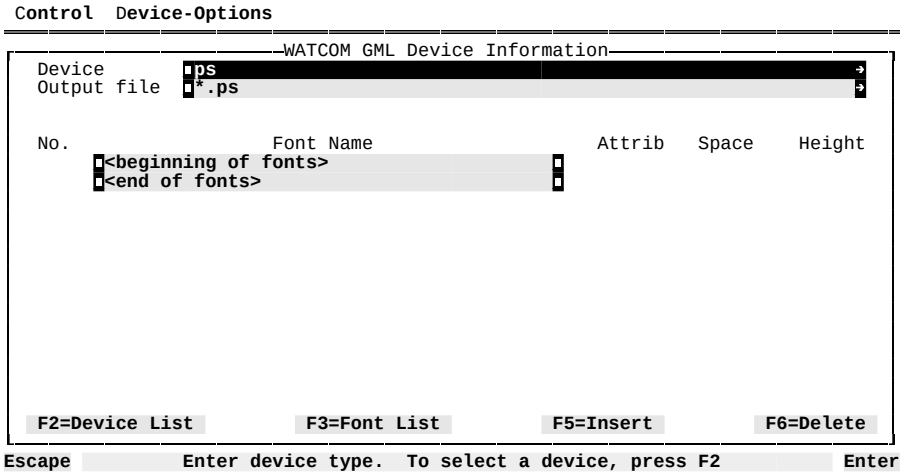

Figure 12. *Device Screen*

The first two areas specify the device name and default output file. The rest of the screen specifies the fonts to be used in the document.

NOTE: The previous screen has no fonts specified in the font area. The fonts numbered zero through three are always defined by the device. If one or all of these fonts are not specified, the device defaults are implied.

Press the **Tab** key and enter `doc.ps`. The output file name will now be 'doc.ps' for all document files processed. Select *Load device defaults* from the *Device-Options* menu.

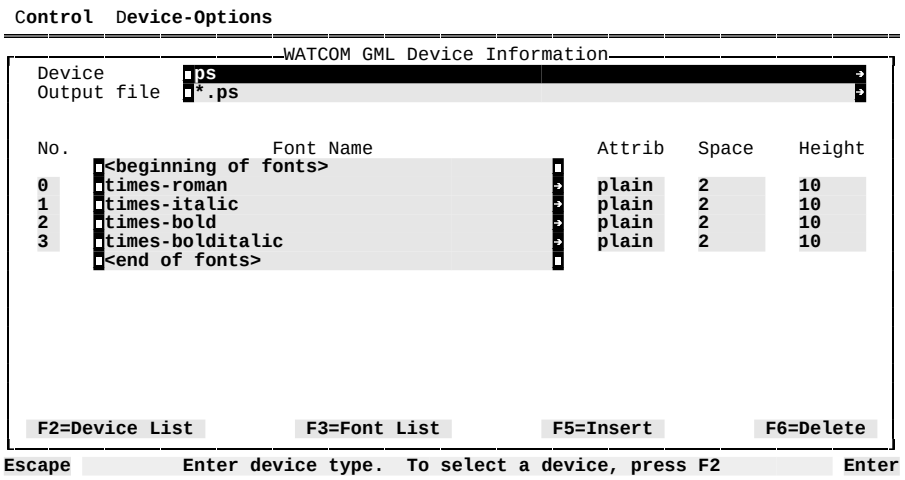

Figure 13. *Default device information*

The value of the output file area will be returned to the default value. The font area is also filled in with the default fonts for the device. Selecting this menu item will delete any existing font definitions and reset the font area to the defaults.

3.2.6.1 Selecting a New Device

You may enter a new device name in the device area or select the *Device List* hotspot. If you select the device list function, a device browser will display a list of available devices.

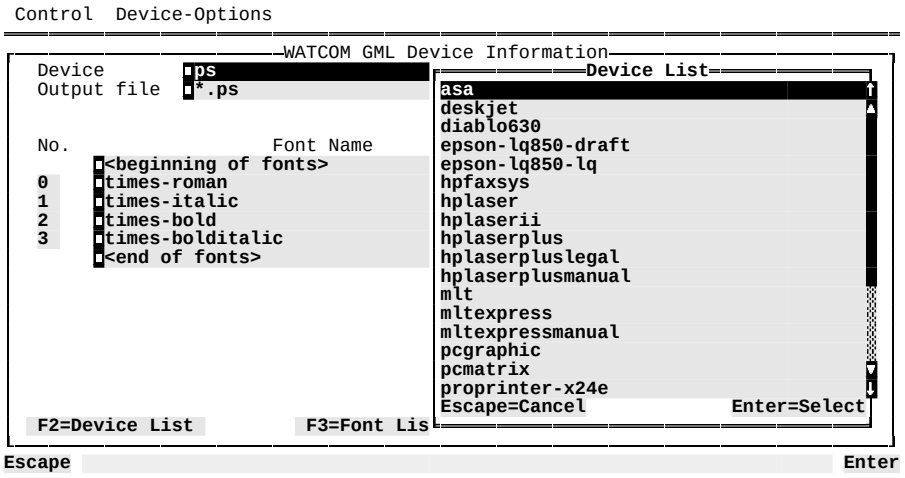

Figure 14. *Device browser*

Mouse:

Move the mouse pointer to the line with 'hplaserplus' and click.

Keyboard:

Press the ↓ key eight times and press the **Enter** key.

The device name HPLASERPLUS will be placed in the device area. You are also asked if you want to set the device defaults. In most cases you will press **Enter** to reset the defaults.

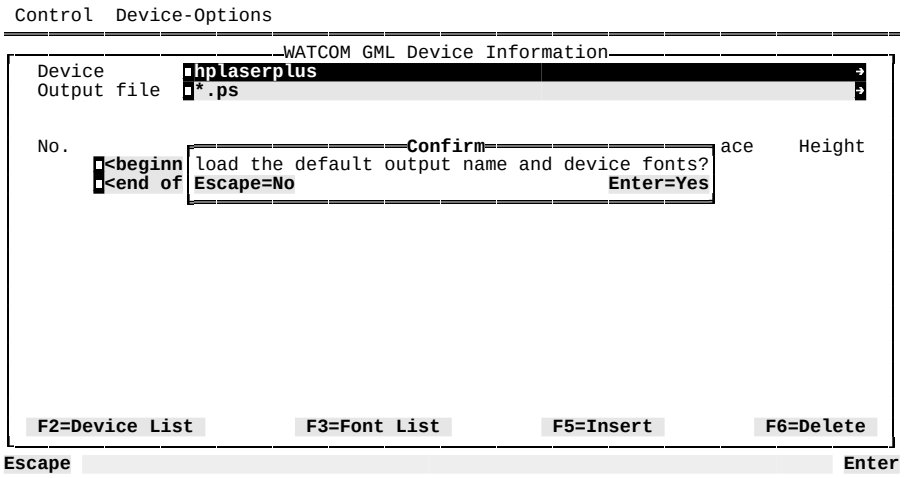

Figure 15. *Selecting a new device*

3.2.6.2 Select Device Fonts

Delete the name 'hplaserplus' from the device area, enter the device name `ps`, and press the **Enter** key. Press **Enter** when asked about loading the defaults. The default fonts for the device will be loaded into the font display area.

Mouse:

Move the mouse pointer to the number area with a value of '1' and click. Select the *Font List* hotspot.

Keyboard:

Press the **Tab** three times followed by the ↓ key. Select the *Font List* hotspot.

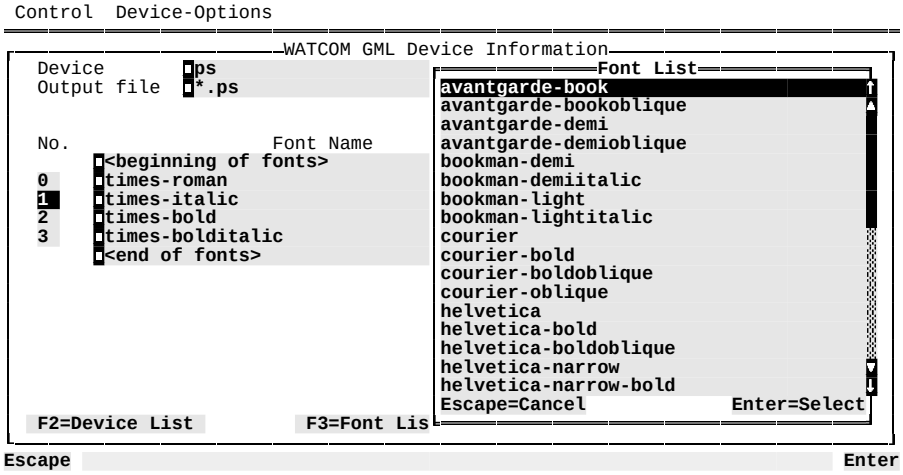

Figure 16. *Available fonts*

The fonts available for the current device are displayed in a font browser.

Mouse:

Move the mouse pointer to the line with 'helvetica' and click.

Keyboard:

Press the ↓ key twelve times and press **Enter**.

The 'helvetica' font will be placed in the font name area, overwriting the previous font name.

Move to the last font line and select the *Insert* hotspot. A new font line area will be created on the screen. Enter the number four (4) and press the **Tab** key. Use the font list function to select a font name for the new area. Press the **Tab** and enter *uline*. This will select the underlining attribute for the font. The value *plain* is usually used when a device has enough variations in the available font style (for example, selecting an italic font instead of underlining a roman font).

The next two columns specify the font *space* and *height* values. These values are numbers with up to two decimal places, and represent point values. A point is a unit of measurement used in typography. WATCOM Script/GML sets 72 points per inch.

The font space defines the amount of space between lines of text which are in the font. If no space was specified, the lines of text would be too close together. Small fonts (1 to 8 points) may require a one point space. Medium fonts (9 to 13 points) may require a two point space. Larger fonts may require three or more points of line space. The amount of line space is adjustable to meet individual requirements. Although this value can be entered for all devices, it is usually applicable to imaging printers such as a laser printer. Line or matrix printers usually have a built in line space value.

The font height defines the height of the font characters. This value may only be specified with *scaleable* fonts. Scaleable fonts are those for which you select a font name and specify its height. Non-scaleable fonts have a built in height value which cannot be adjusted. For example, all of the LaserJet device fonts are non-scaleable.

Enter the numbers two and eleven in the space and height columns. The screen should appear as follows:

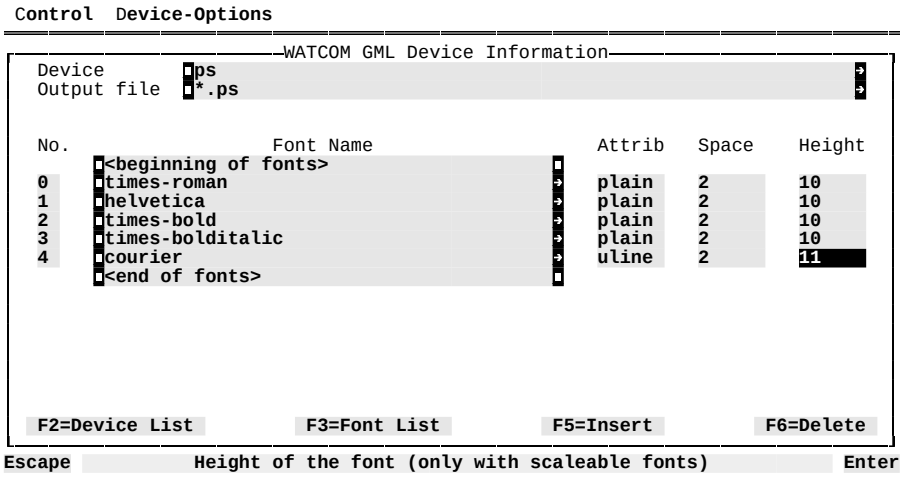

Figure 17. *Define a new font*

Positioning to a font line and selecting the *Delete* hotspot will delete a font from the list. Pressing **Enter** will accept the device and font changes and return you to the options screen. Pressing **Escape** will return you to the options screen without the changes.

3.2.7 Selecting Symbol Information

Symbols may be used in the GML source document to ensure a text fragment is specified the same way in different places. For example,

```
:SET symbol='product' value='WATCOM Script/GML'.
```

will result in the text `WATCOM Script/GML` being placed in the document wherever `&product.` is specified (the period is necessary when specifying a symbol name in the document). The symbol screen is used to define symbol values before processing the document. These values may then be changed without editing the GML source.

From the options screen, select the *Symbol Definitions* hotspot. The symbol screen is organized in a similar manner as the device screen discussed previously. Select the *Insert Symbol* hotspot. Enter the text `product` and press the **Tab** key. Enter the text `WATCOM Script/GML`. The screen should appear as follows:

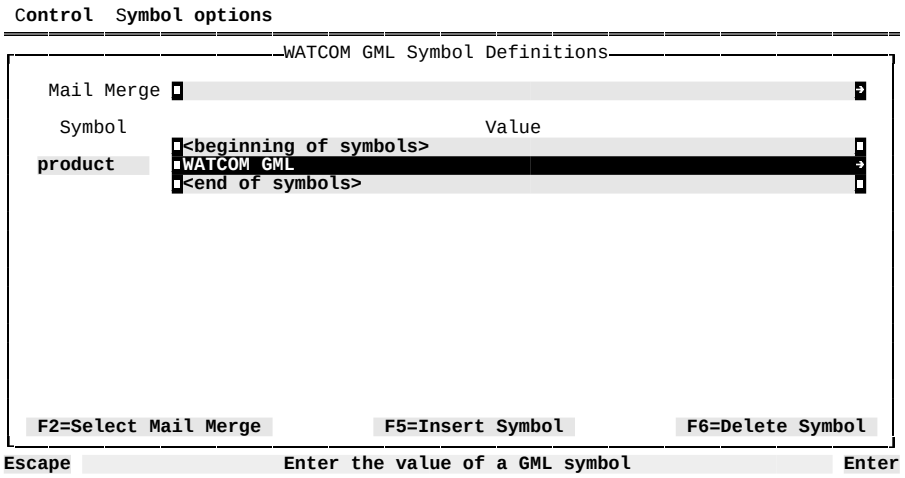

Figure 18. *Define a symbol value*

The symbol name *product* will now have the value `WATCOM Script/GML` if used in the source document.

WATCOM Script/GML has a form letter capability. When a values file containing a list of names and addresses is specified, WATCOM Script/GML will produce one document for each record in the file (although this capability is usually used for producing form letters, it may be used with regular documents as well). The individual values in each record are assigned to pre-defined symbol names (&value1., through to &valuen.). (See "MAILmerge" on page 217 for more information.)

Press the **Tab** twice and enter the text `customer`. The screen should appear as follows:

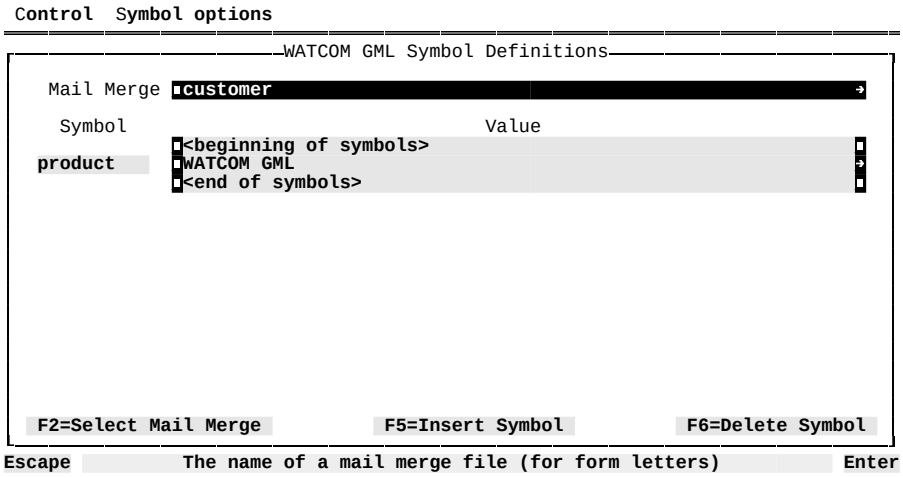

Figure 19. *Define a mail merge file*

3.2.8 Saving Options

After modifying the options and selecting device and symbol information, the document may be processed. However, performing this procedure every time you wish to process the document would be tedious, and likely result in a mistake. Saving the options for future use will allow you to reset all of the options easily.

If you are still in the symbol definitions screen, press **Enter** twice to return to the main document screen. Select *Save current options* from the *Options* menu. The name of the current option file is displayed in the file name area of a browser screen. Enter the name `formletr.opt`. The screen should appear as follows:

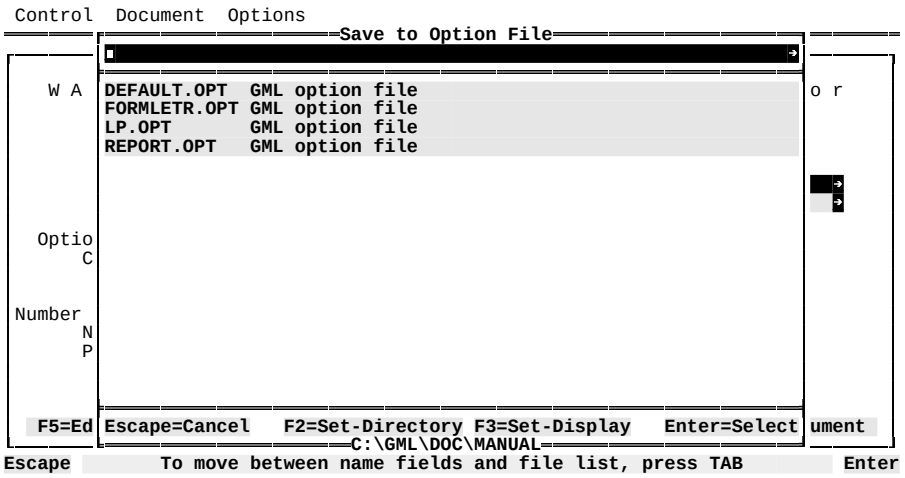

Figure 20. *Save the current options*

Pressing **Enter** will save the options into the specified file.

3.2.8.1 Selecting New Options

Selecting *Select a new set of options* from the *Options* menu will present an options file browser. You may select an existing option file to reset the current options.

3.2.8.2 Default Options

When WATCOM Script/GML starts (both WGMLUI and WGML), the special option file `default.opt` is searched for on the disk. If found, it is loaded before any other options are processed. The default option file may contain small option adjustments from which other option files are based, or contain major changes as demonstrated by the options entered through this tutorial. Placing a `default.opt` file in directories containing similar documents can remove the need to load different option files for different situations. Select *Make current options the default* from the *Options* menu to make the current options your default.

If you have different option files for different documents in one directory, you can load an option file and make it your default when you are going to work on a document over a period of time.

3.2.9 Changing the Configuration

Selecting *Configure* from the *Control* menu presents a dialogue on the screen for changing the defaults used by the WGMLUI program.

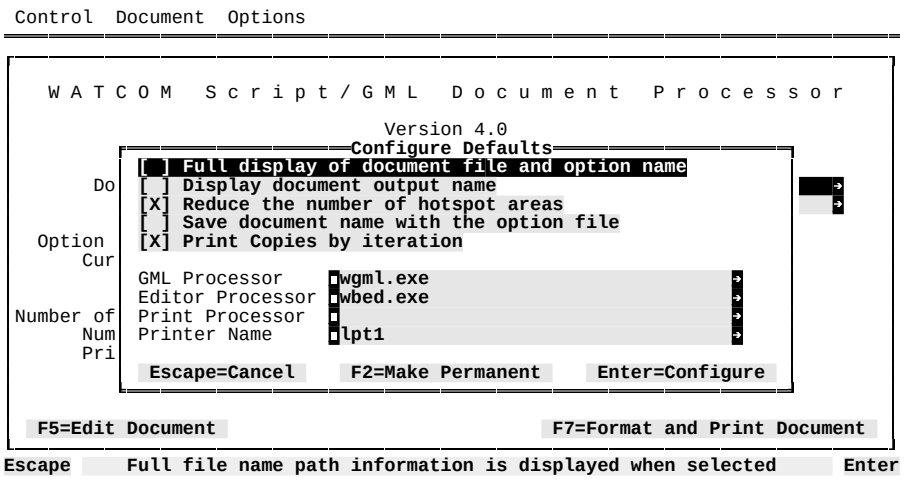

Figure 21. *Configuration Screen*

When you have finished making the configuration changes, pressing the **Enter** key will finish the dialogue. The configuration changes will not be saved when you leave the WGMLUI program. If you wish to make the changes permanent, leave the dialogue by pressing the **F2** key instead.

When options are saved to a file, the first check box in the dialogue specifies if the full document and option file names are to be displayed. If the files are not found in the current directory, the WGMLUI program will redisplay these names with the full path name when this check box is selected.

The second check box will cause the name of the output document file to be displayed on the main document screen. The third check box reduces the number of hotspots on the main document screen. All functions are still available from the menu bar. The fourth check box specifies if the document file name is saved with the options. The fifth check box specifies the method for printing files when more than one copy is requested. If copy iteration is not selected, a print processor program must be specified. The name of the print file, printer port, and the number of copies required are passed as command line parameters to the program. If copy iteration is selected, the print file is processed the number of times specified by the print copies configuration value.

The three processor fields specify programs (or batch files) to run when the *Format*, *Edit File* or *Print* actions are selected. When a print processor is not specified, a copy is performed by the WGMLUI program. The printer name field specifies the device or file name of the printer.

4 Document Elements

4.1 Examples

Text is often included in situations where it is desired that the separate lines in the input are not processed together as they are in a paragraph. One entity for which text is processed like this is called an **example** in the GML language. Consider the following:

```
:GDOC.
:BODY.
:H0.Simple Example
:P.
This is a paragraph which precedes
an example.
Note how WATCOM Script/GML processes the
text in the paragraph.
:XMP.
Lines in an example are
not processed like in a paragraph.


===
:eXMP.
:P.
This text follows the
example and has its words processed in
the same way as the preceding paragraph.
:eGDOC.
```

Figure 22. *Simple XMP example*

The example text which is to be placed in the output is found between the **:xmp** and **:exmp** tags.

The document, when processed, may appear as follows:

```
Simple Example

   This is a paragraph which precedes an
example.   Note  how  WATCOM  Script/GML
processes the text in the paragraph.


  Lines in an example are
  not processed like in a paragraph.


  ===

   This text follows the example and has
its words  processed  in the same way as
the preceding paragraph.
```

***Figure 23.** : Output of* Figure 22

The lines in the example entity have not been processed in the same way as in a paragraph. The layout with which the document is formatted determines aspects such as the indentation and the number of lines skipped before and after the example.

The following tags have been introduced in this section:

:xmp This tags marks the start of an example. Text in an example is not processed in the same way as in a paragraph. With the layout used for this document, the document source lines of a paragraph are processed together and justified. The source lines of an example are not processed together and are not justified.

:exmp This tag marks the end of an example.

Examples always appear in a single column on a page. When there is insufficient room on a page to contain an example, it is placed at the start of the next page.

Once the material in this section is understood, WATCOM Script/GML may be used to format most documents. The remaining features of the language may be viewed as ease-of-use features: the other entities are used to identify specific types of document elements more precisely.

4.2 Notes and Paragraph Continuation

There are other GML entities which process words and input lines in a way that is similar to paragraphs. This section introduces two other entities: paragraph continuation (**:pc**) and note (**:note**).

Paragraph continuation is used when you wish to continue a paragraph which has been interrupted by another GML entity. If the text following the example introduced in the preceding section is a continuation of the paragraph before the example, then the GML source should be altered as follows:

```
:GDOC.
:BODY.
:H0.Simple Example
:P.This is a paragraph which precedes
an example.
Note that words in the paragraph are
processed together.
:XMP.
Lines in an example are
not processed together.
===
:eXMP.
:PC.This text follows the
example and has its input lines processed in
much the same way as a paragraph.
:eGDOC.
```

Figure 24. *Illustration of Paragraph Continuation*

Note that the **:pc** tag has been used instead of the **:p** tag in the text following the example document element.

The document, when processed, may appear as follows:

```
Simple Example

   This is a paragraph which precedes an
example.    Note   that   words  in  the
paragraph are processed together.


  Lines in an example are
  not processed together.
  ===

This text  follows  the  example and has
its input  lines  processed  in much the
same way as a paragraph.
```

***Figure 25.** : Output of* Figure 24

With the layout used for this tutorial, the first line of the text following the example is not indented, causing it to appear as if it were a continuation of the paragraph which precedes the example.

It is tempting to use the **:p** and **:pc** tags interchangeably when it is known that they are processed in an indistinguishable fashion with a given layout. This practice is to be discouraged since it may prevent a document from being formatted properly should the layout be changed.

The paragraph and paragraph continuation tags process input lines together. When the end of an input line is reached, the input text on the next input line is considered to be the start of a new word.

Another GML entity which processes input lines together is a **note**. Consider the following GML specification:

```
:GDOC.
:BODY.
:H0.Simple Note
:NOTE.This is a sample note.
The text within it is processed together.
The words "NOTE: " (or something
similar) precede the text which is
indented.
:eGDOC.
```

***Figure 26.** Illustration of the Note Entity*

The document, when processed, may appear as follows:

32 Notes and Paragraph Continuation

```
Simple Note

NOTE: This is a sample note.    The text
      within  it is processed  together.
      The words  "NOTE: " (or  something
      similar) precede the text which is
      indented.
```

Figure 27. : *Output of* Figure 26

The **:note** tag is used for a block of text that is to be specially noted.

The following tags have been introduced in this section:

:pc	The tag for paragraph continuation is used to indicate the continuation of a paragraph interrupted by another GML entity.
:note	This tag is used for a paragraph which is to be specially noted. The text "Note: " is generated by WATCOM Script/GML.

A number of tags are said to be followed by **paragraph elements**. Paragraph elements are certain GML tags and words of text. The words of text are processed together according to the style dictated by the layout with which the document is being formatted. The **:p**, **:pc** and **:note** tags are all assumed to be followed by paragraph elements.

4.3 Lists

Documents often contain lists. With GML, there are a number of entities that may be used to specify these lists. The entity used depends upon the general type of list. Four of the list types which can be specified in GML are:

Simple	Items in the list are not annotated. A simple list, such as a list of recipe ingredients, do not usually have a particular order.
Unordered	List order is not important. Each item is emphasized by an annotation symbol such as a bullet or asterisk.
Ordered	The list items are annotated by a sequence of numbers. The style of the numbers, such as Roman numerals or Arabic, is determined by the layout.

Definition Each item in the list is annotated by a term that is specified in the GML source. The definitions that you are reading were entered using a definition list.

The content of a list item may be started with a paragraph element. If you do not specify a tag at the beginning of a list item, the text for the list item will be processed together in the same way as with paragraphs. The following subsections illustrate each of these list types.

4.3.1 Simple Lists

Simple lists consist of list items which are displayed in the order entered in the GML source file. The list item is not annotated.

```
:GDOC.
:BODY.
:H0.Illustration of Simple List
:SL.
:LI.This is the first list item.
:LI.This is the second list item.
:LI.This is the last list item.
It is a very long item, in order to
illustrate how text is processed
together.
:eSL.
:eGDOC.
```

Figure 28. *Simple List*

The document, when processed, may appear as follows:

```
Illustration of Simple List

    This is the first list item.

    This is the second list item.

    This is the last list item.  It is a
    very   long   item,   in   order  to
    illustrate  how  text  is  processed
    together.
```

Figure 29. *: Output of* Figure 28

The three list items are presented in blocks separated by a blank line. The blocks are indented from the document text, before and after the list, in a layout-dependent fashion to emphasize that they are list items.

The following tags have been introduced:

:sl	This tag signifies the start of a simple list.
:li	This tag identifies a list item. The text for the list item may be given as a paragraph element.
:esl	This tag signifies the end of a simple list.

4.3.2 Unordered Lists

Unordered lists consist of list items which are displayed in the order given in the GML source. Each item is annotated with some layout-dependent text such as a bullet or asterisk. The presence of the annotation text in the resulting document provides an extra emphasis to each list item.

```
:GDOC.
:BODY.
:H0.Illustration of Unordered List
:UL.
:LI.This is the first list item.
:LI.This is the second list item
in the unordered list.
:P.This is a paragraph
in the second list item.
:LI.This is the last list item.
It is a very long item, in order to
illustrate how text is processed
together.
:eUL.
:eGDOC.
```

Figure 30. *Unordered List*

The document, when processed, may appear as follows:

```
Illustration of Unordered List

*   This is the first list item.

*   This is the  second list item in the
    unordered list.

       This is a paragraph in the second
    list item.

*   This is the last list item.  It is a
    very   long   item,   in   order  to
    illustrate  how  text  is  processed
    together.
```

***Figure 31.** : Output of* Figure 30

Each list item is preceded by the annotation symbol. The second item consists of two paragraphs. The text of the list items has been indented in a layout-dependent fashion.

The following tags have been discussed:

:ul	This tag signifies the start of an unordered list.
:li	This tag identifies a list item. The text for the list item may be given as a paragraph element.
:eul	This tag signifies the end of an unordered list.

4.3.3 Ordered Lists

Ordered lists consist of list items which are displayed in the order given in the GML source. Each item is annotated in a sequence. The annotation distinguishes each list item and is often used to itemize steps in a procedure.

```
:GDOC.
:BODY.
:H0.Illustration of Ordered List
:OL.
:LI.This is the first list item.
:LI.This is the second list item
in the ordered list.
:P.This is a paragraph
in the second list item.
:LI.This is the last list item.
It is a very long item, in order to
illustrate how text is processed
together.
:eOL.
:eGDOC.
```

Figure 32. *Ordered List*

The document, when processed, may appear as follows:

```
Illustration of Ordered List

1.  This is the first list item.

2.  This is the  second list item in the
    ordered list.

       This is a paragraph in the second
    list item.

3.  This is the last list item.  It is a
    very   long   item,   in   order  to
    illustrate  how  text  is  processed
    together.
```

Figure 33. *: Output of* Figure 32

The three list items are each preceded by a list item number. The second item consists of two paragraphs. The text of the list items has been indented in a layout-dependent fashion.

The following tags have been discussed:

:ol	This tag signifies the start of an ordered list.
:li	This tag identifies a list item. The text for the list item may be given as a paragraph element.
:eol	This tag signifies the end of an ordered list.

4.3.4 Definition Lists

Definition lists are similar to the other lists, except that the annotation text which is to precede a list item is supplied by a **:dt** (**D**efinition **T**erm) tag. The contents of the list item follows a **:dd** (**D**efinition **D**escription) tag. List items may contain a number of paragraphs.

```
:GDOC.
:BODY.
:H0.Illustration of Definition List
:DL.
:DT.Term-1
:DD.This is the first list item.
:DT.Term-2
:DD.This is the second list item
in the definition list.
:P.This is the second paragraph
of the second list item.
:DT.Last Term
:DD.This is the last list item.
It is a very long item, in order to
illustrate how text is processed
together.
:eDL.
:eGDOC.
```

Figure 34. *Definition List*

The document, when processed, may appear as follows:

```
Illustration of Definition List

Term-1    This is the first list item.

Term-2    This is the  second  list item
          in the definition list.

             This    is    the    second
          paragraph  of the second  list
          item.

Last Term This  is the last  list  item.
          It  is a  very  long  item, in
          order to  illustrate  how text
          is processed together.
```

Figure 35. *: Output of* Figure 34

The three list items are each preceded by the term given in the **:dt** tag. The second item consists of two paragraphs. The text of the list items has been indented in a layout-dependent fashion.

The following tags have been discussed:

:dl	This tag signifies the start of a definition list.
:dt	This tag specifies the term to precede the text for the list item. The text of the definition term is given as a **text line**.
:dd	This tag identifies the start of the text for a list item. The text for the list item may be given as a paragraph element.
:edl	This tag signifies the end of a definition list.

4.3.5 Nesting lists

A list can appear as part of a list item. The "inner" list is said to be **nested** inside the outer list. This is illustrated by the following:

```
:GDOC.
:BODY.
:H0.Illustration of Nested Lists.
:OL.
:LI.Outer level -- item(1)
:LI.Outer level -- item(2)
:OL.
:LI.Inner level -- item(1)
:LI.Inner level -- item(2)
:eOL.
:LI.Outer level -- item(3)
:LI.Outer level -- item(4)
:eOL.
:eGDOC.
```

Figure 36. *Illustration of Nested List*

The document, when processed, may appear as follows:

```
Illustration of Nested Lists.

1.  Outer level -- item(1)

2.  Outer level -- item(2)

    i)  Inner level -- item(1)

    ii) Inner level -- item(2)

3.  Outer level -- item(3)

4.  Outer level -- item(4)
```

***Figure 37.** : Output of* Figure 36

Note that an ordered list is nested inside the second item of the outside ordered list. All list types may be nested as part of another list item. The nesting can take place to arbitrary depth, although excessive nesting will tend to make the document hard to read.

4.3.6 List Parts

A list part is a special entity which may be included to temporarily suspend a list. This entity is used to provide an explanation for the list items which follow. The list part entity causes the indentation to be reset to the value at the start of the current list. A paragraph element is then formatted at this indentation until the next list item is encountered with a **:li** or **:dt** tag. Consider the following example:

```
:GDOC.
:BODY.
:H0.Illustration of List Part
:OL.
:LI.This is item(1).
:LI.This is item(2).
:LP.This is the list part.
Note that it contains paragraph elements
with the input lines processed together.
:P.This is the second paragraph of the
list part.
:LI.This is item(3).
:LI.This is item(4).
:eOL.
:eGDOC.
```

***Figure 38.** Illustration List Part*

The document, when processed, may appear as follows:

```
Illustration of List Part

1.  This is item(1).

2.  This is item(2).

This  is the list  part.   Note  that it
contains  paragraph  elements  with  the
input lines processed together.

   This is the second  paragraph  of the
list part.

3.  This is item(3).

4.  This is item(4).
```

Figure 39. *: Output of* Figure 38

The list part in the example consists of two paragraphs. The annotation of the list items is continued following the list part. List parts may be used with any of the list types.

5 Document Structure

To this point in the tutorial, the general structure of a document has been simplified as summarized in the following figure.

```
:GDOC.
  :BODY.
    . . . body of document
:eGDOC.
```

Figure 40. *Simplified Document Structure*

The complete structure of a document may be summarized as follows.

```
:GDOC.
  :FRONTM.
    . . . front material
  :BODY.
    . . . main body of document
  :APPENDIX.
    . . . appendices
  :BACKM.
    . . . back material
:eGDOC.
```

Figure 41. *Overall Structure of Document*

As illustrated in Figure 41, there are four major segments with the following contents:

Front Material This segment contains entities such as the title page, abstract, preface, table of contents and list of figures.

Body This segment contains the main text for the document.

Appendix This segment contains the appendices for the document.

Back Material This segment contains any ending text which follows the appendices, such as the index.

Each of the four segments is optional. The complexity of a document determines which segments will be used.

5.1 Headings

The GML language considers a document to be composed of major sections with further levels of subdivision. GML has seven levels of document division. These divisions are indicated with the heading tags(:h0, :h1, ..., :h6).

The following restrictions apply to the use of headings:

abstract **:h0** and **:h1** tags may not be used.

preface **:h0** and **:h1** tags may not be used.

body All heading tags may be used.

appendix **:h0** tags may not be used.

back material **:h0** tags may not be used.

Consider the following example:

```
:GDOC.
:BODY.
:H0.Arithmetic Primer
 :H1.Addition
  :H2.Adding two digits
  :H2.Adding multiple digits
 :H1.Subtraction
  :H2.Subtracting two digits
   :H3.Positive Results
   :H3.Negative Results
  :H2.Subtracting multiple digits
   :H3.Borrowing not required
   :H3.Borrowing required
    :H4.Positive Results
    :H4.Negative Results
:H0.Arithmetic Reference
 :H1.Numbers
 :H1.Addition
 :H1.Subtraction
:eGDOC.
```

Figure 42. *Sample Headings*

The preceding example shows the structure of a fictitious document. One or more document elements could have been specified following any of the **:h0**, **:h1**, **:h2 :h3** or **:h4** tags. A common convention is to have the heading tags represent the following entities:

:h0	major parts of a document
:h1	chapters or appendices
:h2	sections
:hn (n<7)	subsections of sections defined with **:h(n-1)** tags.

With an appropriate layout, the following table of contents would be produced by the example:

```
Arithmetic Primer

1 Addition

  1.1 Adding two digits
  1.2 Adding multiple digits

2 Subtraction

  2.1 Subtracting two digits
    2.1.1 Positive Results
    2.1.2 Negative Results
  2.2 Subtracting multiple digits
    2.2.1 Borrowing not required
    2.2.2 Borrowing required
      2.2.2.1 Positive Results
      2.2.2.2 Negative Results

Arithmetic Reference

1 Numbers

2 Addition

3 Subtraction
```

Figure 43. *Sample Table of Contents*

Each heading level can be formatted in a different way. The level zero headings have not been numbered, while the heading levels one through four have been numbered as directed by the layout.

The layout with which the document is formatted determines the format of headings. Some of the formatting actions determined by the layout are the number of lines skipped before a heading is displayed and whether the heading at a particular level is to be included in the table of contents. The GML tag used to obtain a table of contents will be discussed later in the tutorial.

The heading tags are immediately followed by the words which make up the heading. This heading text is called a **text line** in the reference section of this manual. When a text line is presumed to follow a tag, the text is usually given on the same line, immediately following the period(.) after the tag. The paragraph tag, **:p**, introduced earlier in the tutorial does not have a text line associated with it; the text which follows the tag is used to form a paragraph, and may be given on any number of lines.

5.2 Front Material

The front material starts with the **:frontm** tag and may contain up to five entities: title page (**:titlep**), abstract (**:abstract**), preface (**:preface**), table of contents (**:toc**) and list of figures (**:figlist**). The particular entities to be included will depend upon the requirements of a document. When none of the five entities are necessary, the entire front material segment may be omitted from the document. The following subsections discuss each of the entities in the front material.

5.2.1 Title Page

A title page specification begins with a **:titlep** tag and concludes with an **:etitlep** tag. Between these two tags the document title (**:title**), the document number (**:docnum**), the date of publication (**:date**) and the author(s) (**:author**) with their address(es) (**:address**) may be specified. These tags may be given in any order; the **:author**, **:title** and **:address** tags may be specified a number of times. The layout with which a document is formatted establishes the style used to prepare the title page.

A title page is illustrated in the following example:

```
:GDOC.
 :FRONTM.
   :TITLEP.
     :TITLE.My Summer Vacation
     :DOCNUM.765
     :DATE.September 25, 1985
     :AUTHOR.J.W.Welch
     :AUTHOR.B.J.Welch
     :ADDRESS.
       :ALINE.University of Waterloo
       :ALINE.Waterloo, Ontario, Canada
     :eADDRESS.
   :eTITLEP.
:eGDOC.
```

Figure 44. *Sample Title Page*

The document, when processed, may appear as follows:

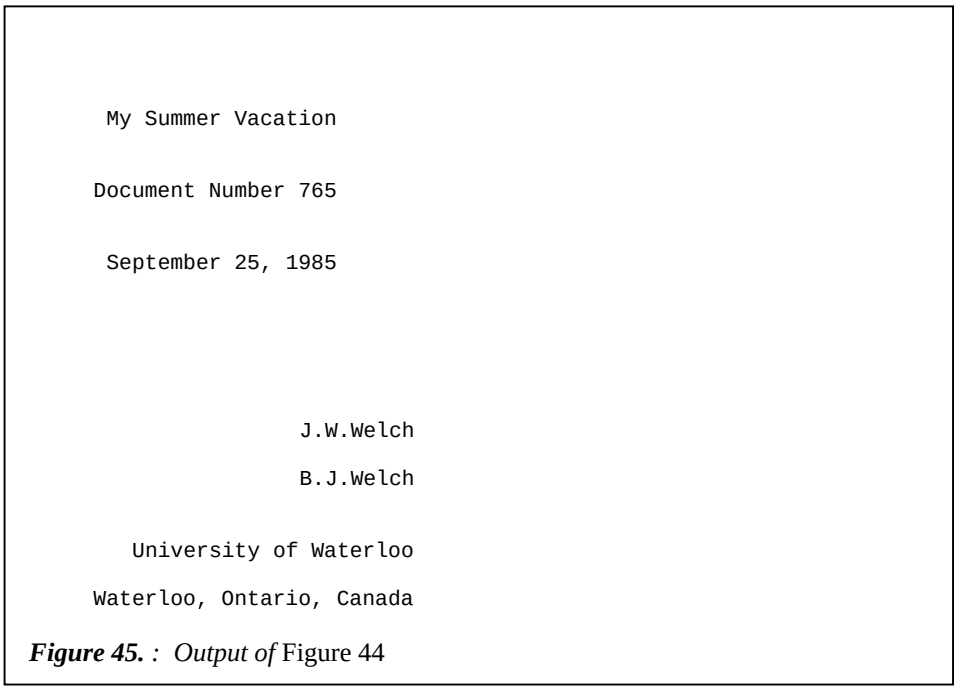

Figure 45. *: Output of* Figure 44

The following tags have been introduced in this section:

:titlep	This tag indicates the start of the title page specification.
:etitlep	This tag indicates the end of the title page specification.
:title	This tag is used to give the title of the document, and may be specified more than once for mulitiple title lines.
:docnum	This tag is used to specify the document number.
:date	This tag is used to specify the date printed on the document. The date on which the document is processed is used when date text is not specified with the date tag.
:author	This tag may be used to specify the name of an author. It may be specified a number of times in the title page.
:address	This tag is used to specify the start of an address entity. The entity consists of a number of address lines given by **:aline** tags.

:eaddress	This tag is used to indicate the end of an address entity.
:aline	This tag specifies an address line in an address entity, and may be specified more than once.

5.2.2 Abstract

An abstract may be specified following the **:abstract** tag. Heading tags (except **:h0** and **:h1**) may be used within the abstract. The abstract section is implicitly terminated by the next major document section.

5.2.3 Preface

A preface may be specified following the **:preface** tag. Heading tags (except **:h0** and **:h1**) may be used within the preface. The preface section is implicitly terminated by the next major document section.

5.2.4 Table of Contents

The inclusion of the **:toc** tag causes the table of contents to be printed. The GML source files of some documents are processed more than once before they are actually printed. In this case, the table of contents is placed in the front material of the document. When the source files of the GML document are processed only once, the table of contents cannot be placed in the front material. For a one pass document where a table of contents is requested, WATCOM Script/GML will create the table of contents at the end of the document. Documents are processed more than once when WATCOM Script/GML is instructed to use a number of passes (see "PASSes" on page 219).

5.2.5 List of Figures

The **:figlist** tag causes a list of figures to be created. The figure list is processed in the same way as the table of contents when dealing with single versus multiple pass documents.

5.3 Body

Most documents will contain a main body, which starts after the **:body** tag. The complexity of the document determines how many levels of headings should be used.

A simple document may contain very few headings. A major document, such as this book, may contain major parts and numerous subsections.

5.4 Appendices

The appendix segment of a document occurs following the **:appendix** tag. The appendix segment is very similar to the body segment, except that **:h0** tags may not be specified. The usual convention is to use **:h1** tags to name the appendices and to use higher-numbered heading tags for the sections and subsections within an appendix.

5.5 Back Material

The back material consists of a text portion followed by the index, and is started with the **:backm** tag. The text and index are both optional.

The text portion is intended for the specification of any closing comments (such as an epilogue) which pertains to a document. The text area is similar to that of the body, except that **:h0** tags may not be specified.

The **:index** tag causes an index to be printed as part of the back material. The index command line option must be specified for the index to be created (see "INDex/NOINDex" on page 216).

5.6 Large Documents

It is often convenient when preparing large documents to use more than one GML source file. For example, each chapter in a book may be a separate GML file. The entire book is prepared by causing WATCOM Script/GML to process a primary input file containing **:include** tags which cause the individual chapters to be included into the document.

```
:GDOC.
:BODY.
 :INCLUDE file='chap1'.
 :INCLUDE file='chap2'.
 :INCLUDE file='chap3'.
:APPENDIX.
 :INCLUDE file='app1'.
 :INCLUDE file='app2'.
:eGDOC.
```

Figure 46. *Illustration of :include Tag*

The preceding figure illustrates one such organization for a book. The **:include** tag has been used to include the text for three chapters and two appendices. Replacing each **:include** tag with the contents of the referenced file would produce the same document.

The **:include** tag causes WATCOM Script/GML to process the associated file at the place the tag occurs. An included file may itself cause other files to be included. This capability allows the author to organize the document into separate data files. Using a number of files is important in complex documents since a number of small files is usually easier to maintain than would be the case if the entire document were specified as a single large file.

When a document is organized into small files, a subset of the input files may be included into a "skeleton" file containing the major tags (such as **:gdoc.**, **:body** and **:egdoc**). The subset of input files may then be processed independently from the rest of the document as it is being written and revised. This capability is useful for a number of reasons:

1. WATCOM Script/GML will format a small document in less time than would be required for a large document.

2. Several authors may work simultaneously on independent parts of a document.

3. WATCOM Script/GML runs on a number of different machines. A document can be prepared on a micro-computer in sections, and then transferred to a mainframe computer for final production of the document.

Consequently, most documents of a non-trivial nature are specified using a number of input files.

The **:include** tag is not a standard tag; it has been added to the tags accepted by WATCOM Script/GML to provide the capability to include GML source files. Other GML processors may use alternative tags or methods to include additional files.

6 Additional Document Elements

6.1 Highlighting Phrases

The GML language provides a mechanism to highlight phrases to emphasize fragments of text. Four levels of highlighting may be specified:

Highlight level(0)
Highlight level(1)
Highlight level(2)
Highlight level(3)

The levels are illustrated in the preceding list. This list was created with the following GML input:

```
:SL.
:LI.:HP0.Highlight level(0):eHP0.
:LI.:HP1.Highlight level(1):eHP1.
:LI.:HP2.Highlight level(2):eHP2.
:LI.:HP3.Highlight level(3):eHP3.
:eSL.
```

Figure 47. *Illustration of Highlight tags*

The highlighting is started with a **:hpn** (n = 0, 1, 2, 3) tag and terminated with a corresponding **:ehpn** tag. The formatted result of a particular highlighting tag is determined when the document is processed. The type of output device for which the document is being prepared and the character sets being used are important factors in determining the formatted result. In the preceding simple list, the formatted result is a mixture of the different character sets which are used to produce this manual. After the GML source has been entered, the selection of the output device or character sets can be changed until the formatted output is produced as desired.

Most text, such as the text which composes a paragraph, is at highlighting level zero. Some document elements, such as an example, may have a layout-determined highlighting level different from that of the basic text of the document. The **:hp0** tag is therefore a means of ensuring that the highlighting level of zero is used. The other main use of the **:hp0** tag is to obtain the highlighting level of zero while in the midst of other highlighting levels.

It should be noted that the highlight phrase tags may be used with a portion of a word. The GML specification

```
:HP2.G:eHP2.eneralized
```

will cause the word "**G**eneralized" to be output, with the first letter emphasized with the level two highlighting.

Highlight tags may not occur with tags having text lines for which the GML layout determines the highlighting. Examples of such tags include **:title**, **:dt** and the heading tags.

When a tag ends a sentence, make sure that you use two periods. The first period will be processed as part of the tag. The second period will be treated as part of the text to be processed. If only one period is specified, the period to end the sentence will not appear in the resulting document.

```
:GDOC.
:BODY.
:H0.Tag at the end of a Sentence
:P.
Care must be taken with a tag at
the end of a :HP0.sentence:eHP0..
Note that the previous sentence is
specified with two periods, but
only one is used as text.
:eGDOC.
```

Figure 48. *Tag at the end of a Sentence*

The document, when processed, may appear as follows:

```
Tag at the end of a Sentence

   Care  must be taken with a tag at the
end  of  a  sentence.   Note   that  the
previous  sentence is specified with two
periods, but only one is used as text.
```

Figure 49. *: Output of* Figure 48

6.2 Citations

Citations are used to refer to titles of books. Although the highlighting tags could be used to emphasize the title, the use of the citation entity means that WATCOM Script/GML can process the title in a different manner than the regular highlighting tags. The following figure illustrates a citation:

```
:P.
The GML language is described in
:CIT.WATCOM Script/GML Tutorial
and Reference Manual:eCIT..
```

Figure 50. *Illustration of a Citation*

A citation starts with a **:cit** tag and is completed by an **:ecit** tag. For each **:cit** tag there must be a corresponding **:ecit** tag. Note that there are two periods after the **:ecit** tag. The first period ends the tag. The second period is text to end the sentence.

The same restrictions apply to citations as were given with highlighting; citations should not be given where the GML layout determines the emphasis given to a phrase.

6.3 Quotations

GML provides two types of quotations: short quotations (inline quotations) and long quotations (or excerpts). Short quotations are used to quote a few phrases or sentences within a block of text; long quotations are used to identify a block of text which is a quote.

The following example illustrates a short quotation:

```
:GDOC.
:BODY.
:H0.Short Quotation
:P.Wes has often said to us,
:Q.Never do a project within another project.:eQ.
:P.We have often replied,
:Q.We need the second project to accomplish the
first.
You wouldn't want us to operate
:Q.in a technological vacuum:eQ.,
would you?:eQ.
:eGDOC.
```

Figure 51. *Illustration of a Short Quotation*

The document, when processed, may appear as follows:

```
Short Quotation

   Wes has often said to us, "Never do a
project within another project."

   We  have often  replied, "We need the
second  project to accomplish the first.
You  wouldn't  want us to  operate 'in a
technological vacuum', would you?"
```

***Figure 52.** : Output of* Figure 51

Note that the first quoted phrase, enclosed by the **:q** and **:eq** tags, has been placed within quotation characters. The second quoted phrase has another quoted phrase nested inside it. Note the nested phrase is emphasized with apostrophe (') characters. The use of short quotations is encouraged, where applicable, for a number of reasons:

1. It encourages uniformity.

2. Different quotation styles may be used depending upon the layout used and the device which is used to print the document.

A short quotation may be used whenever text is being processed.

The following example illustrates a long quotation:

```
:GDOC.
:BODY.
:H0.Long Quotation
:P.This will serve as an example
of a long quotation.
Suppose the following came from a memo:
:LQ.
:P.Wes has often said to us,
:Q.Never do a project within another
project.:eQ.
:P.We have often replied,
:Q.We need the second project to
accomplish the first.
You wouldn’t want us to operate
:Q.in a technological vacuum:eQ.,
would you?:eQ.
:eLQ.
:PC.That sums it pretty well.
:eGDOC.
```

Figure 53. *Illustration of a Long Quotation*

The document, when processed, may appear as follows:

```
Long Quotation

   This  will  serve as an  example of a
long quotation.    Suppose the following
came from a memo:

     Wes has  often said to us, "Never
  do   a   project    within   another
  project."

     We  have often  replied, "We need
  the second project to accomplish the
  first.   You  wouldn’t  want  us  to
  operate ’in a technological vacuum’,
  would you?"

That sums it pretty well.
```

Figure 54. *: Output of* Figure 53

It may be noted that the text of the preceding example, enclosed by the **:lq** and **:elq** tags, has been indented according to the style of the layout used. A long quotation will cause the implicit end of a paragraph entity.

6.4 Figures

Figures are used to create space for illustrated material to be placed in the document. If you enter text within the figure, it is processed in the same manner as the input text in an example.

A figure is illustrated in the following example:

```
:GDOC.
:BODY.
:H0.Simple Figure
:P.This is a paragraph which precedes
a figure.
Note that words in the paragraph are
processed together.
:FIG.
Lines in a figure are
not processed together.
===
:eFIG.
:P.This text follows the
figure and has its words processed
together.
:eGDOC.
```

Figure 55. *Very Simple Figure*

The document, when processed, may appear as follows:

```
Simple Figure

   This is a  paragraph which precedes a
figure.    Note   that   words   in  the
paragraph are processed together.


Lines in a figure are
not processed together.
===

   This text  follows the figure and has
its words processed together.
```

Figure 56. *: Output of* Figure 55

The figure begins with the **:fig** tag and is completed by the **:efig** tag. The lines between these tags are not processed together.

Most figures will have captions. The caption is supplied by the **:figcap** tag as illustrated in the following example:

```
:GDOC.
:BODY.
:H0.Illustration of the Figure Caption
:P.This is a paragraph which precedes
a figure.
Note that words in the paragraph are
processed together.
:FIG.
Lines in a figure are
not processed together.
===
:FIGCAP.Illustration of Caption
:eFIG.
:P.This text follows the
figure and has its words processed
together.
:eGDOC.
```

Figure 57. *Figure with Caption*

The document, when processed, may appear as follows:

```
Illustration of the Figure Caption

   This is a  paragraph which precedes a
figure.    Note   that   words   in  the
paragraph are processed together.


Lines in a figure are
not processed together.
===

Figure 1. Illustration of Caption

   This text  follows the figure and has
its words processed together.
```

Figure 58. *: Output of* Figure 57

Note that the caption is included in the figure and that WATCOM Script/GML automatically supplied a number for the figure. When a figure list is created in the front material, the figure caption is used to create a figure list entry.

A description of the figure may be included following the caption. This is illustrated as follows:

```
:GDOC.
:BODY.
:H0.Illustration of the Figure Description
:P.This is a paragraph which precedes
a figure.
Note that words in the paragraph are
processed together.
:FIG.
Lines in a figure are
not processed together.
===
:FIGCAP.Illustration of a Caption
:FIGDESC.This description illustrates
how a description may be associated
with the caption of a figure.
:eFIG.
:P.This text follows the
figure and has its words processed
together.
:eGDOC.
```

Figure 59. *Illustration of a Description with Figure*

The description, specified with the **:figdesc** tag, follows the caption.

```
Illustration of the Figure Description

   This is a  paragraph which precedes a
figure.    Note   that   words   in  the
paragraph are processed together.


Lines in a figure are
not processed together.
===

Figure 1. Illustration   of  a  Caption:
          This  description  illustrates
          how  a   description   may  be
          associated with the caption of
          a figure.

   This text  follows the figure and has
its words processed together.
```

Figure 60. *: Output of* Figure 59

The text associated with the description is processed together following the caption. WATCOM Script/GML automatically supplied a colon after the caption to separate it from the description.

6.5 Referencing

A reference is used to direct the reader to a specific place in the document for more information. During the creation and revision of a document, the location of the referenced entity may shift, making it difficult to correctly maintain such reference information as the page number. The GML reference tags automate this function by creating the reference for you. This section will illustrate the use of simple referencing with headings and figures.

When a GML entity is referenced, there must be a way to uniquely identify the entity being referenced. This is done with the use of an **attribute**. Attributes modify the action of the tag or supply additional information, such as an identifier name. All attributes are entered before the period which ends the tag.

The **id** attribute allows you to assign a unique identifier to an entity. The following example illustrates the use of the **id** attribute:

```
:GDOC.
:BODY.
:H0 id='firsth0'.Illustrate ID with a Heading
:FIG id='myfig'.
This figure has
an ID attribute
:FIGCAP.
:eFIG.
:eGDOC.
```

Figure 61. *Illustration of the ID Attribute*

The text *firsth0* is the value of the heading **id** attribute. This value is assigned to the heading, and must be unique within the document. The text *myfig* is the identifier name for the figure. When a figure is assigned an identifier, a figure caption must also be specified.

A reference to a heading or figure can be made with the **:hdref** and **:figref** tags respectively. Both of these tags may appear anywhere in your input text, and require the presence of the **refid** attribute. This attribute is used to reference a particular entity. The following example illustrates the use of the two referencing tags:

```
:GDOC.
:BODY.
:H0 id='firsth0'.Illustrate ID with a Heading
:P.
In this section we deal with some
arbitrary topic.
:FIG id='myfig'.
This figure has
an ID attribute
:FIGCAP.
:eFIG.
:H0.New Section
:P.
Now we are in a new part of the document.
At this point, we can say:
For more information,
see :HDREF refid='firsth0'.
and :FIGREF refid='myfig'..
The heading text will be inserted for the
heading reference, and the figure number
will be used with the figure reference.
:eGDOC.
```

Figure 62. *Illustration of Referencing*

The document, when processed, may appear as follows:

```
Illustrate ID with a Heading

   In  this  section  we  deal with some
arbitrary topic.


This figure has
an ID attribute

Figure 1.

New Section

   Now  we  are  in a  new  part  of the
document.   At this  point,  we can say:
For more information, see "Illustrate ID
with  a  Heading"  and  Figure  1.   The
heading  text will be  inserted  for the
heading reference, and the figure number
will be used with the figure reference.
```

Figure 63. *: Output of* Figure 62

The attribute value for the **refid** attribute is the identifier name of the entity we wish to reference. Note that there are two periods after the figure reference. The first period ends the tag, while the second period is text to end the sentence.

The text *"Illustrate ID with a Heading"* is inserted where the heading reference tag was specified. The text *Figure 1* is inserted where the figure reference tag was specified. If the reference was on a different page than the referenced entity, the page number of the referenced entity is also inserted. For example, if the figure reference was on page 12 and the referenced figure was on page 9, the text to be inserted would be *Figure 1 on page 9*.

Referencing of headings allows you to change the wording of a heading at any time and still have the correct heading text used when you reference the heading in other parts of the document. Figure referencing removes any dependence upon the ordering of the figures in the document.

6.6 Indexing

GML will create an index for you using information gathered during the processing of the document. The index information is supplied by index tags. This section will illustrate the use of the indexing tags to create a simple index.

With a large document, working drafts can be produced faster without the index. To process the index information, the INDEX option (see "INDex/NOINDex" on page 216) must be specified on the WATCOM Script/GML command line.

When an index tag is specified, the text line following the tag and the current page number are saved. If the **:index** tag is specified in the back material, and the INDEX option is specified, the saved information is processed and output as an index. The following example illustrates the use of the index tags:

```
:GDOC.
:BODY.
:I1.primary index
:I2.primary index subentry
:I2.another subentry
:I3.subentry of an I2 entry
:BACKM.
:INDEX.
:eGDOC.
```

Figure 64. *Illustration of the Indexing Tags*

The document, when processed, may appear as follows:

```
   +---+
   | P |
   +---+

   primary index  1

      another subentry  1
         subentry of an I2 entry  1
      primary index subentry  1
```

Figure 65. *: Output of* Figure 64

The **:i1** tag causes the creation of a primary index entry. The **:i2** tags cause the creation of index subentries for the last primary index entry. The **:i3** tag causes the creation of index subentries for the last level two index entry. Note that WATCOM Script/GML automatically sorted the index subentries. All index entries are sorted when the index is created.

If an index entry is specified more than once, the entries are merged. The following example illustrates index entry merging:

```
:GDOC.
:BODY.
:H0.Start of the document
:I1.primary index
:I2.primary index subentry
:H0.More of the same document
:I1.second primary
:I2.subentry
:I3.subentry of an I2
:I2.subentry
:I1.primary index
:BACKM.
:INDEX.
:eGDOC.
```

Figure 66. *A More Complex Index*

The document, when processed, may appear as follows:

```
   +---+
   | P |
   +---+

   primary index  1-2

      primary index subentry  1


   +---+
   | S |
   +---+

   second primary  2

      subentry  2
         subentry of an I2  2
```

***Figure 67.** : Output of* Figure 66

The two index subentries created by the **:I2** tags with the text "subentry" were merged together. Note that since they were both for the same output page, the page number was only displayed once. The level one index entries with the text "primary index" were also merged together. However, since they were for different output pages, the page number for each entry was displayed.

A different method to create an index entry is with the index heading tag. The index heading tags generate results similar to the index tags described above. The main difference is that the page number is not saved. The following example illustrates the index heading tags:

```
:GDOC.
:BODY.
:IH1.primary index
:I2.primary index subentry
:I1.second primary
:I2.subentry
:IH1.primary index
:I2.another subentry
:BACKM.
:INDEX.
:eGDOC.
```

***Figure 68.** Illustration of Index Headings*

The document, when processed, may appear as follows:

```
   +---+
   | P |
   +---+

   primary index

      another subentry  1
      primary index subentry  1


   +---+
   | S |
   +---+

   second primary  1

      subentry  1
```

***Figure 69.** : Output of* Figure 68

Note that the level one index with the text "primary index" does not have any page numbers displayed. It can also be seen that the subentry for the second **:ih1** tag is merged with the subentry for the first **:ih1** tag.

7 Layouts

Information about the document style is not specified when a GML document file is created. Style information such as indentation on the first line of a paragraph does not change the paragraph into a different type of document entity. Examples of information considered part of the document style are:

1. The maximum number of lines on a page.
2. The number of spaces to indent the first line in a paragraph.
3. The maximum number of characters on a line, including the space characters.
4. The number of lines to leave between paragraphs.
5. Justify text by adding space between words.

With WATCOM Script/GML, the document style is specified in the **layout** section. A document is produced by associating a layout with the document source when it is processed by WATCOM Script/GML. By specifying different layouts, the document style can be changed without modifying the document text.

NOTE: The layout determines how many columns of text are on an output page. Widows and most document elements are placed on the next available column when the current column is full. For the purposes of this book, a one column layout is assumed.

7.1 Specifying a Layout

Initial values for all of the layout items are defined in WATCOM Script/GML, and is called the **default layout**. A new layout is created by modifying the initial values of the default layout. Only those values that you wish to change need to be specified.

The layout section starts with the **:layout** tag and ends with the **:elayout** tag. The layout values are grouped into sections, and are specified in the same way as the GML tags. The sections are identified by layout tags, with the individual layout values specified by tag attributes. Most of the GML document elements have corresponding layout tags.

The layout section is specified before the **:gdoc** tag. The best way to do this is with the **layout** option on the WATCOM Script/GML command line (see "LAYout" on page 216). You may also choose to include the layout specification directly into your source document. Specifying the layout on the command line makes it easier to switch layouts.

The layout section may be specified more than once. Authors can share a common layout and still specify layout changes for their own document. Each layout section modifies the values defined by the default layout plus the cumulative modifications of previous layout sections.

7.2 General Modifications

Some of the layout items control general aspects of the document instead of specific document elements. One of these is the **:page** layout tag.

```
:LAYOUT
:PAGE
        left_margin = '.5i'
        right_margin = 80
:eLAYOUT.
:GDOC.
   .
   .
   .
```

The **:page** layout tag defines information about the page you are printing on. The attributes used in the example above define the left and right margins. The other attributes of this tag do not have to be specified if you do not need to change their values.

The *right_margin* attribute value is eighty(80) characters. The actual amount of space that this value represents depends on the size of the characters used to produce the document. The *left_margin* attribute is half of one inch, and does not have any dependency on the character set used. For more information on the possible width values, see "Horizontal Space Unit" on page 76.

7.3 Modifying Document Elements

Many of the tags in the WATCOM Script/GML layout relate directly to the tags used in specifying the document. The following example illustrates a modification to the way in which the example tag is formatted:

```
:LAYOUT
:XMP
        pre_skip = 3
        post_skip = 2
:eLAYOUT.
```

The attributes *pre_skip* and *post_skip* define the amount of space to leave before (pre) and after (post) the example tag. The attribute values are in line units, which means the amount of space will depend on the height of a text line within an example. For more information on the possible vertical space values, see "Vertical Space Unit" on page 77.

The skip attributes imply certain actions. Skip values which are specified as line units are multiplied by the current spacing value. For example, if the example is double spaced, the skip value will be doubled. Skips are also merged. If an example was to follow another example, the amount of skip between them would be three.

7.4 Obtaining the Current Layout

The current layout definitions may be obtained by converting the definitions into a text file.

```
:LAYOUT.
:CONVERT file='currlay'.
:eLAYOUT.
```

The file CURRLAY.GML will be created by processing the above text with WATCOM Script/GML. Since no other layout sections are specified, the produced information will be the default layout which is built into WATCOM Script/GML.

7.5 Banners

With WATCOM Script/GML, banners define the text content at the top and/or bottom of the output page. Banners are also called running titles and running footers. Most of the default layout consists of banner definitions.

The most common use of a banner is to define the place on the output page to display the current heading and page number. The following illustrates what might appear at the bottom of a document page.

```
WATCOM Script/GML      3
```

The current heading starts at the left hand side of the page, and the current page number is set to the right hand side of the page. The banner therefore consists of two distinct

types of information. Banners are subdivided into regions which contain different types of information. The regions for the above example would look as follows:

```
+------------------------+
| WATCOM Script/GML  | 3 |
+------------------------+
```

The banner starts with the **:banner** tag, and defines the size and placement of the banner. Within the banner definition, each region is defined with the **:banregion** tag. Since each region contains only one type of information, the banner region tag and its attributes must be specified for each area of the banner.

7.5.1 Defining the Banner

The banner tag has a number of attributes which define the area of the output page to place the banner. The following shows how to define the banner area for our previous example.

```
:BANNER
        left_adjust = 0
        right_adjust = 0
        depth = 3
        docsect = body
        place = botodd
```

The *left_adjust* and *right_adjust* attributes allow you to change the banner margins relative to the page margins. The value of the left adjust attribute is added to the left margin of the page. The value of the right adjust attribute is subtracted from the page right margin.

The *depth* attribute specifies the depth of the entire banner. The banner regions can be placed through more than one output line, or placed above one another.

The *docsect* attribute determines the document section in which the banner will be used. In this example, the value BODY means that the banner will appear in the body section of the document. If the value HEAD0 was specified, the banner would only be used when a heading of level zero appears on the output page.

The *place* specifies the page position of the banner. The value BOTODD means the bottom of odd numbered pages. The values that can be specified are:

```
top       top of all pages
topodd    top of odd pages
topeven   top of even pages
bottom    bottom of all pages
botodd    bottom of odd pages
boteven   bottom of even pages
```

7.5.2 Defining the Banner Region

The banner region tag has a number of attributes which define the area of the banner in which the region is placed. The following shows how to define one of the banner regions for our previous example.

```
:BANREGION
        indent = 0
        hoffset = left
        width = extend
        voffset = 2
        depth = 1
        font = 0
        refnum = 1
        region_position = left
        pouring = last
        contents = headtext0
:eBANREGION.
```

The horizontal position of the banner region within the banner is defined by the *hoffset* and *indent* attributes. The horizontal offset attribute (hoffset) defines the offset from the left edge of the banner. In this case, the value LEFT specifies that the region should start at the left side of the banner. The indent attribute supplies an indentation to be applied to the region position after the horizontal offset is determined.

The *width* attribute defines the width of the banner. The value EXTEND sets the region width to be from the start point to the right edge of the banner or the next region, whichever comes first. Use this value for regions with a large width. If the page margins change, the region widths will automatically be adjusted for the new margins.

The *voffset* attribute defines the vertical offset from the top of the banner area to the top of the region. If the banner depth is "3", a vertical offset of "2" will start the region on the last line of the banner. This would put the text on the last line of the page and ensure that at least two blank lines are left between the main body of text and the banner text.

The *depth* attribute defines the depth of the region within the banner. If it specifies more than one line space, text which does not fit on the first line of the region will be split to the following region lines.

Each region must be uniquely identified by the value of the *refnum* attribute. These values are later used if one region of a banner is replaced. The *font* attribute defines the character set used in the region.

Within the banner region, text can be placed with the *region_position* attribute. The value LEFT specifies that the text should be placed starting at the left side of the region.

The *pouring* attribute is used to specify how headings are placed in a banner. If the requested heading does not appear on the page, the value LAST obtains the last heading used in the document of the same level (in the previous example, this would be level zero). This attribute is ignored if the region contents does not contain heading information.

The *contents* attribute specifies the content of the banner region. The value HEADTEXT0 requests the text component of the last heading zero produced in the document. This value would not include the heading number. A number of different values may be specified with this attribute, including constant string data.

7.5.3 Sample Banner Definition

The following shows the banner definition which would create the bottom banner for the example output shown earlier.

```
:BANNER
        left_adjust = 0
        right_adjust = 0
        depth = 3
        docsect = body
        place = topodd
:BANREGION
        indent = 0
        hoffset = left
        width = extend
        voffset = 2
        depth = 1
        font = 0
        refnum = 1
        region_position = left
        pouring = last
        contents = headtext0
:eBANREGION.
:BANREGION
        indent = 0
        hoffset = right
        width = 3
        voffset = 2
        depth = 1
        font = 0
        refnum = 2
        region_position = right
        pouring = none
        contents = pgnuma
:eBANREGION.
:eBANNER.
```

7.5.4 Using Symbols in Banner Definitions

Symbol names may be specified in the string value of a banner definition content attribute. A number of symbol names are defined when a banner is created to contain special values. To create a bottom banner with the page number centered and surrounded by dashes as shown in the following,

```
+------------------------+
|          - 1 -         |
+------------------------+
```

create the following banners:

```
:BANNER
        left_adjust = 0
        right_adjust = 0
        depth = 3
        place = bottom
        docsect = body
:BANREGION
        indent = 0
        hoffset = left
        width = extend
        voffset = 2
        depth = 1
        font = 0
        refnum = 1
        region_position = center
        pouring = last
        contents = '- &amp.$pgnuma. -'
:eBANREGION
:eBANNER
```

GML Reference

8 General Specifications

8.1 Processing Rules

WATCOM Script/GML processes the source document text in a particular sequence. Each input record is divided into smaller *logical* records, each containing a specific type of information. The following rules are applied to the input record in sequence.

1. The input record is searched for GML tags. The input record is split into a new logical record at each GML tag. The one exception to this rule is the :CMT tag which results in the entire input line, including other GML tags, to be treated as a comment and not processed. Recognition of a GML tag in the text may be defeated by using the &GML. symbol instead of the GML tag separator.

2. As each logical record is needed for processing, substitution of symbols is performed.

If the SCRIPT or WSCRIPT option has been specified on the command line.

3. If the value of a symbol starts with the Script control word separator (default of ';'), the input record is split into two logical records and substitution stops. The separator character does not appear in either logical record.

4. If the first character of a logical record is the Script control word indicator (default of '.'), the record must be a Script control word or macro line. The control word indicator will be recognized if it is the first character in a symbol value being substituted at the beginning of the record.

5. When a Script control line is specified, the list of defined macros is searched. If a macro with the given name is not found, the value must be a Script control word.

6. If a Script control line is specified with a second control word indicator (.. at the beginning of the logical record), the list of defined macros is not searched.

7. If a Script control line is specified with an apostrophe after the control word indicator, control word separators are not recognized in the logical record.

8. Control word separators in a Script control line will cause a split into a new logical record at that point and stop symbol substitution. The separator character does not appear in either logical record.

NOTE: When there is more than one pass over the document source, the layout section is only processed on the first pass.

8.2 Horizontal Space Unit

The term **horizontal space unit** is used throughout this document to indicate the use of one of the following forms of measurement:

Centimeter	The number of centimeters followed by the **CM** symbol. The number may have up to two decimal digits specified. Example: **5.23CM**
Characters	The number of characters. The width of a character is determined by the CPINCH command line option. The default option for this value is 10 characters per inch. Example: **23**
Cicero	The number of ciceros followed by the **C** symbol and up to two didot point digits. There are twelve didot points in a cicero, with 72 points in an inch. Example: **9C9**
Device Units	The number of characters. The width of the character zero (0) in the current font is used. Example: **23DV**
Ems	The number of ems followed by the **M** symbol. The width of an em space is the width of the character 'M' in the current font. Example: **9M**
Inch	The number of inches followed by the **I** symbol. The number may have up to two decimal digits specified. Example: **1.25I**
Millimeter	The number of millimeters followed by the **MM** symbol. The number may have up to two decimal digits specified. Example: **25.75MM**

Pica — The number of picas followed by the **P** symbol and up to two point digits. There are twelve points in a pica, with 72 points in an inch. Example: **6P12**

To prevent a period from indicating the end of the tag, single quotes may be used to enclose the space unit value. Since enclosing the space value with quotes is always correct, it is best to use them in all cases.

8.3 Vertical Space Unit

A vertical space unit is specified in the same way as a horizontal space unit. An EM space specifies the number of lines, the height of a line determined by the current font, adjusted for the document spacing value currently in effect. For example, a vertical space value of '2M' with double spacing in effect results in four lines worth of space.

An integer number specifies the number of lines, the height of a line determined by the LPINCH command line option, adjusted for the document spacing value currently in effect. The default lines per inch value is 6.

A device unit space(DV) specifies the number of lines without the current document spacing accounted for. For example, a vertical space value of '2DV' with double spacing in effect results in two lines worth of space.

8.4 Font Linkage

Attributes in the GML tag set and the layout which accept vertical or horizontal space values are linked to specific fonts. When values are specified in terms of characters (such as 12 or 5M), the absolute amount of space is determined using a font assigned to that attribute value. All values not explicitly linked to a font use the default font.

8.5 Tag Attributes

A tag attribute is used to modify or define the behaviour of the tag. For example, the **depth='5i'** attribute is used with the figure tag to specify the depth of the figure. This attribute will reserve five inches of space in the document for pasting in a figure. The value of the attribute (ie '5i') must not be split across input records.

Attribute values may be enclosed in quotes. Either single or double quotes may be used. If the value contains a quote character which is the same as the enclosing quotes,

the quote can be specified twice to enter it into the value. Accents are also accepted as a quoting character. A character string must be enclosed in quotes.

The attributes of a tag may be specified over a number of input records, and are separated from the tag and each other by a space. Although the attributes are separated from the tag, they are considered part of the tag specification and must precede the period which ends the tag. Attribute values which contain a period should be enclosed in quotes to prevent the termination of the tag specification.

8.6 Symbolic Substitution

A symbol is a name which represents an arbitrary string of text. Once a symbol is assigned a text value, the symbol can be used in the document source in place of that text. Consider the following:

```
:SET symbol='product'
     value='WATCOM Script/GML'.
:GDOC.
:BODY.
:P.
Symbolic substitution is quite
simple with &product..
:eGDOC.
```

Figure 70. *Symbolic Substitution*

The document, when processed, may appear as follows:

```
   Symbolic substitution is quite simple
with WATCOM Script/GML.
```

Figure 71. *: Output of* Figure 70

A symbol name is defined and assigned a string of text with the **:set** tag (see "SET" on page 107). The value of the symbol name can be defined at any point in the document file. Any valid character string may be assigned to the symbol name. When the symbol is referenced later, the value is substituted into the input text. The substitution is done before the source text or input translation is processed by WATCOM Script/GML.

A symbol name is preceded by an ampersand(&) when referenced, and is terminated by any character not valid in a symbol name. If the terminating character is a period, it is considered part of the symbol specification (you must therefore remember to specify

two periods if a symbol ends a sentence). The recognition of a symbol name is case insensitive.

The symbol name should not have a length greater than ten characters, and may only contain letters, numbers, and the characters @, #, $ and underscore(_). Specifying the letters SYS as the first three characters of the symbol name is equivalent to specifying a dollar($) sign.

Recursive substitution is performed on a symbol. This means that the text substituted for a symbol is checked for the presence of more symbol names. As well, if the symbol name is immediately followed by another symbol name (no intervening period or blanks), new names can be constructed from the successive substitutions. For example:

```
:SET symbol='prodgml'
     value='WATCOM Script/GML'.
:SET symbol='prodname'
     value='gml'.
:GDOC.
:BODY.
:P.
Symbolic substitution is quite
simple with &prod&prodname...
:eGDOC.
```

Figure 72. *Iterative Substitution*

The first part of the symbol sequence, *&prod*, does not exist as a defined symbol. However, when *&prodname.* is substituted, the resulting symbol name *&prodgml* exists. The resulting substitution produces the following:

```
   Symbolic substitution is quite simple
with WATCOM Script/GML.
```

Figure 73. *: Output of* Figure 72

If an asterisk is specified immediately before the symbol name (ie symbol='*prodname' or &*prodname.), then the symbol is local. Local symbols may not be referenced outside the file or macro in which they are defined. If an undefined local symbol is referenced in a macro, it is replaced with an empty value.

8.7 Identifiers

Identifiers are used to "identify" certain types of document elements so that they may be referenced. For example, identifiers are useful with headings. If an identifier is assigned to a heading with the *id* attribute, the heading can be referenced with the **:hdref** tag. The heading to be referenced is "identified" by the identifier name assigned to the heading. If the heading text is later changed, the heading reference will still be valid, and automatically use the new heading text.

An identifier name should not be longer than seven characters and must consist of letters and numbers. If the identifier name is longer than seven characters, a warning message will be issued.

8.8 Input Translation

Some of the characters available with a particular output device may not be characters that can be entered into the input text. Input translation provides a way to enter this type of data. A special escape character may be selected in the layout. (See "DEFAULT" on page 144). If this escape character is entered into the GML input text, the character immediately following it will be translated to the value specified as the input translation value for that character. Most characters are defined to be unchanged by input translation. Consider the following:

```
:GDOC.
:BODY.
:P.
Input translation is useful for
entering characters not available
on the keyboard, such as the
bullet (/*) character.
It can also be used to prevent
the normal space(/ ) expansion.
:eGDOC.
```

Figure 74. *Input Translation*

If the input translation escape character is the slash(/), the processed document may appear as follows:

```
   Input   translation   is  useful  for
entering characters not available on the
keyboard,   such   as  the   bullet  (*)
character.   It  can  also  be  used  to
prevent the normal space( ) expansion.
```

***Figure 75.** : Output of* Figure 70

Note that the bullet character in the output is the asterisk. The example output for this manual was produced with the terminal device, which does not have special characters. If the value '**/*'** is used within the text of this document, the character '•' is produced.

Input translation is performed when text is separated into words. The translated character is not examined during these operations, providing a method for bypassing the normal processing rules of WATCOM Script/GML. The values which result from an input translation are defined in the device character sets (see "InTrans Block" on page 245).

9 GML Tags

This chapter contains a subsection on each of the tags supported by the WATCOM Script/GML language. The tags are presented in alphabetical order and in several forms:

1. tag.

   This form is used when other data is not associated with the tag. For example, the **:body** tag is used when defining the structure of the document, but has no text specified with it.

2. tag.<paragraph elements>

   This form is used when paragraph elements, such as text, are assumed to follow the tag. The **:pc** tag is an example of this type of tag.

3. tag.<text line>

   Some tags have a single line of text associated with it, such as with the **:h0** tag. In this situation, the processing rules are as follows:

   1. When nothing follows the tag, except an optional period(.), the next input line is used.

   2. Otherwise, the text following the period or space after the tag is used.

   As these rules are somewhat complicated, it is best to always place the text line on the same line as the tag, immediately following a period.

Use two periods when a tag ends a sentence. The first period will be processed as part of the tag. The second period will be treated as part of the text. If only one period is specified, the period to end the sentence will not appear in the resulting document.

Basic document elements, such as highlighting tags, cannot appear as part of a <text line>. If a tag is specified, it will be processed as if it had been entered on the next input record.

Some tags have attributes which are used to modify or define the behavior of the tag. Tag attributes will be presented in the following way:

```
:tag  attribute-one
      [attribute-two]
```

Attributes not enclosed in brackets([), such as attribute-one, are required and must be present with the tag. Most attributes are optional, and will have brackets as illustrated by attribute-two. No other text is allowed between a tag and its attributes.

9.1 ABSTRACT

```
Format: :ABSTRACT.
```

This tag signals the start of the abstract in the front material of a GML document. The abstract is optional, but must follow the title page section if specified. Basic document elements and heading levels two through six (:h2-:h6) may be specified in the abstract.

9.2 ADDRESS

```
Format: :ADDRESS.
```

An address entity may be used as a basic document element or as part of the title page. Each line of the address entity is specified by the **:aline** tag. A corresponding **:eaddress** tag must be specified for each **:address** tag.

9.3 ALINE

```
Format: :ALINE.<text>
```

The address line tag specifies a line in an address entity (**:address**). The **:aline** tag is specified for each line of a multiple line address.

9.4 APPENDIX

```
Format: :APPENDIX.
```

The appendix section is optional, and follows the body section of a GML document. The appendix causes an implicit end to the body section, and may contain basic document elements and heading levels one through six (:h1-:h6).

9.5 AUTHOR

```
Format: :AUTHOR.<text line>
```

This tag specifies the name of an author, and may only appear within the title page specification. The **:author** tag is specified for each author when there is more than one.

9.6 BACKM

```
Format: :BACKM.
```

The back material is the last section in a GML document. Basic document elements, and heading levels one through six (:h1-:h6) and the **:index** tag may be specified. The index tag must be the last tag specified in the back material.

9.7 BINCLUDE

```
Format: :BINCLUDE file='file name'
                  depth='vert-space-unit'
                  reposition=start
                             end.
```

The binary include tag causes the data in the specified file to be included into the document without being processed by WATCOM Script/GML. This tag provides the means to include graphic or non-textual data in the document (see also "GRAPHIC" on page 96).

The required attribute **file** specifies the name of the file to include. The value of the attribute is a character string, and may be any valid file name. The input file is processed as containing binary data. If the input is text data, a record type such as "(t:80)" must be prefixed to the file name (see "Files" on page 281).

The required attribute **depth** specifies the vertical size of the contents of the file. The value of the attribute is any non-zero vertical space unit. This depth value must be the exact depth of the file contents when placed in the formatted output, and is used to reserve the required amount of space on the page. The depth attribute is linked to the current font being used in the document (see "Font Linkage" on page 77).

The required attribute **reposition** specifies the place in the formatted output that new text would be placed after the content of the file is processed. With some devices, a graphic will not change the current position on the output page when it is processed. In this case, WATCOM Script/GML must ensure that following text is started on the output page after the graphic. The attribute value *start* indicates that the current

position on the output page will be unchanged after the graphic is processed. The attribute value *end* indicates that the current position on the output page will be immediately following the graphic after the graphic is processed. If the included data is within a framed figure, and the frame is formed with characters, the value of the reposition attribute must be the value *start*.

9.8 BODY

```
Format: :BODY.
```

This tag signals the start of the main body of a GML document. The document body is composed of headings and basic document elements.

9.9 CIT

```
Format: :CIT.
```

This tag starts the highlighting of a citation (e.g., the title of a book). The actual highlighting to be performed is determined by the layout and the type of output device the document is processed for. Examples of highlighting include underlining, displaying in bold face, or using a different character shape (such as italics).

A citation may not be used where the GML layout explicitly determines the emphasis to be used, such as in the text of a heading.

The citation tag is a paragraph element. It is used with text to create the content of a basic document element, such as a paragraph. A corresponding **:ecit** tag must be specified for each **:cit** tag.

9.10 CMT

```
Format: :CMT.
```

The information following the comment tag on the input line is treated as a comment. Text data and GML tags following the comment tag are not processed. Except between tag attributes, this tag may appear at any point in the GML source.

9.11 DATE

```
Format: :DATE.<text line>
```

The date tag appears in the title page specification. The current date is used if the optional date text line is not specified. If the date text line is specified, the entered text is used in other parts of the document when the date is required.

9.12 DD

```
Format: :DD.<paragraph elements>
            <basic document elements>
```

This tag signals the start of the text for an item description in a definition list. The definition description tag must be preceded by a corresponding **:dt** tag, and may only appear in a definition list.

9.13 DDHD

```
Format: :DDHD.<text line>
```

The definition description heading tag is used to specify a heading for the definition description of a definition list. It must be preceded by a corresponding **:dthd** tag, and may only appear in a definition list. The heading tag may be used more than once within a single definition list.

9.14 DL

```
Format: :DL [compact]
            [break]
            [headhi=head-highlight]
            [termhi=term-highlight]
            [tsize='hor-space-unit'].
```

The definition list tag signals the start of a definition list. Each list item in a definition list has two parts. The first part is the definition term and is defined with the **:dt** tag. The second part is the definition description and is defined with the **:dd** tag. A corresponding **:edl** tag must be specified for each **:dl** tag.

The **compact** attribute indicates that the list items should be compacted. Blank lines that are normally placed between the list items will be suppressed. The compact attribute is one of the few WATCOM Script/GML attributes which does not have an attribute value associated with it.

The **break** attribute indicates that the definition description should be started on a new output line if the size of the definition term exceeds the maximum horizontal space normally allowed for it. If this attribute is not specified, the definition description will

be placed after the definition term. The break attribute is one of the few WATCOM Script/GML attributes which does not have an attribute value associated with it.

The **headhi** attribute allows you to set the highlighting level of the definition list headings. Non-negative integer numbers are valid highlighting values.

The **termhi** attribute allows you to set the highlighting level of the definition term. Non-negative integer numbers are valid highlighting values.

The **tsize** attribute allows you to set the minimum horizontal space taken by the definition term. Any valid horizontal space unit may be specified. The attribute value is linked to the font of the :DT tag if the termhi attribute is not specified (see "Font Linkage" on page 77).

9.15 DOCNUM

```
Format: :DOCNUM.<text line>
```

This document number tag appears in the title page specification, and specifies the number associated with the document. The default text "Document Number " is generated before the text line.

9.16 DT

```
Format: :DT.<text line>
```

This tag is used to specify the term which is defined for each item in a definition list. It is always followed by a **:dd** tag, which specifies the start of the text to define the term, and may only appear in a definition list.

9.17 DTHD

```
Format: :DTHD.<text line>
```

The definition term heading tag is used to specify a heading for the definition terms of a definition list. It is always followed by a **:ddhd** tag, and may only appear in a definition list. The heading tag may be used more than once within a single definition list.

9.18 EADDRESS

```
Format: :eADDRESS.
```

This tag signals the end of an address entity. An address entity may be used as a basic document element or as part of the title page. A corresponding **:address** tag must be previously specified for each **:eaddress** tag.

9.19 ECIT

```
Format: :eCIT.
```

This tag ends the highlighting of a citation. A corresponding **:cit** tag must be previously specified for each **:ecit** tag.

9.20 EDL

```
Format: :eDL.
```

This tag signals the end of a definition list. A corresponding **:dl** tag must be previously specified for each **:edl** tag.

9.21 EFIG

```
Format: :eFIG.
```

This tag signals the end of a figure. A corresponding **:fig** tag must be previously specified for each **:efig** tag.

9.22 EFN

```
Format: :eFN.
```

This tag signals the end of a footnote. A corresponding **:fn** tag must be previously specified for each **:efn** tag.

9.23 EGDOC

```
Format: :eGDOC.
```

This tag signals the end of a GML document. It must be the last tag specified in the input source. A corresponding **:gdoc** tag must be specified at the beginning of the document.

9.24 EGL

```
Format: :eGL.
```

This tag signals the end of a glossary list. A corresponding **:gl** tag must be previously specified for each **:egl** tag.

9.25 EHP0, EHP1, EHP2, EHP3

```
Format: :eHPn.
        (n=0,1,2,3)
```

These tags end the highlighting of phrases at one of the four levels provided by GML. Each **:ehpn** tag must be preceded by a corresponding **:hpn** tag.

9.26 ELAYOUT

```
Format: :eLAYOUT.
```

This tag signals the end of a layout section. A corresponding **:layout** tag must be previously specified for each **:elayout** tag.

9.27 ELQ

```
Format: :eLQ.
```

This tag signals the end of a long quote. A corresponding **:lq** tag must be previously specified for each **:elq** tag.

9.28 EOL

```
Format: :eOL.
```

This tag signals the end of an ordered list. A corresponding **:ol** tag must be previously specified for each **:eol** tag.

9.29 EPSC

```
Format: :ePSC.
```

This tag signals the end of a process specific control section. A corresponding **:psc** tag must be previously specified for each **:epsc** tag.

9.30 EQ

```
Format: :eQ.
```

This tag signals the end of a quote. A corresponding **:q** tag must be previously specified for each **:eq** tag.

9.31 ESF

```
Format: :eSF.
```

This tag ends the highlighting of phrases started by the last **:sf** tag.

9.32 ESL

```
Format: :eSL.
```

This tag signals the end of a simple list. A corresponding **:sl** tag must be previously specified for each **:esl** tag.

9.33 ETITLEP

```
Format: :eTITLEP.
```

This tag signals the end of the GML document title page. A corresponding **:titlep** tag must be previously specified for the **:etitlep** tag.

9.34 EUL

```
Format: :eUL.
```

This tag signals the end of an unordered list. A corresponding **:ul** tag must be previously specified for each **:eul** tag.

9.35 EXMP

```
Format: :eXMP.
```

This tag signals the end of an example. A corresponding **:xmp** tag must be previously specified for each **:exmp** tag.

9.36 FIG

```
Format: :FIG [depth='vert-space-unit']
             [frame=box
                    rule
                    none
                    'character string']
             [id='id-name']
             [place=top
                    bottom
                    inline]
             [width=page
                    column
                    'hor-space-unit'].
             <paragraph elements>
             <basic document elements>
```

This tag signals the start of a figure. Each line of source text following the figure tag is placed in the output document without normal text processing. Spacing between words is preserved, and the input text is not right justified. Input source lines which do not fit on a line in the output document are split into two lines on a character, rather than a word basis. A figure may be used where a basic document element is permitted, except within a figure, footnote, or example.

If the figure does not fit on the current page or column, it is forced to the next one. If the current column is empty, the figure will be split into two parts.

The **depth** attribute accepts vertical space units as possible values. The amount of specified vertical space is created in the output before any source input text is processed. The value of the depth attribute is linked to the current font (see "Font Linkage" on page 77).

The **frame** attribute will determine the framing value for the figure. The layout for the document specifies a default frame value if the frame attribute is not specified. The frame is created with the appropriate characters for the output device selected. The attribute value *box* will cause the entire figure to be enclosed by a box. The attribute

value *rule* will cause a line to be created before the top and after the bottom of the figure. The sides of the figure will not be enclosed. The rule line at the top of the figure is not produced if the place of the figure is *top*. The bottom rule is not produced if the place of the figure is *bottom*. The attribute value *none* will cause no framing to occur. If a character string is used as the framing value, a framing value of rule will be in effect, using the specified character string to create the rule lines.

The **id** attribute will associate an identifier name to the figure. If an identifier name is specified, the figure caption tag (**:figcap**) must also be specified in the figure. The quoted name is used by the **:figref** tag to generate a figure reference to the figure.

The **place** attribute determines the page position of the figure. The layout for the document specifies a default place value if the place attribute is not specified. A place value of *top* causes the figure to be placed at the top of the next available page or column. Any text which follows the figure in the input may 'float' before the figure to fill up the previous page. A place value of *bottom* causes the figure to be placed at the bottom of the next available page or column. Any text which follows the figure in the input may float before the figure to fill up the previous page and before the figure on the current page. A place value of *inline* causes the figure to be output within the context of the input text which surrounds it. Text which follows the figure in the input will not float before the figure.

The **width** attribute allows you to specify the width of the figure. The attribute value *page* specifies that the figure will be as wide as the page, even if the document is formatted for more than one column. The attribute value *column* specifies that the figure shall be one column wide. If a horizontal space unit is used as the attribute value, the figure will have the width specified by the attribute value. The width attribute value is linked to the font of the figure (see "Font Linkage" on page 77).

9.37 FIGCAP

```
Format: :FIGCAP.<text line>
```

The figure caption tag is used within a figure to specify the caption for the figure. The figure caption tag must be specified if the figure has an identifier name associated with it. Layout defined text followed by the figure number and a delimiter is inserted before the caption text (the default text and delimiter is "Figure" and a period). The figure caption follows the main text of the figure.

9.38 FIGDESC

```
Format: :FIGDESC.<paragraph elements>
                 <basic document elements>
```

This tag signals the start of the description for a figure. The tag is placed after the optional **:figcap** tag within a figure. The GML processor automatically adds a colon(:) following the caption when a figure description is present.

9.39 FIGLIST

```
Format: :FIGLIST.
```

This tag may be used in the front material of a GML document to request that the list of figures be formatted. More than one pass will be required to create the figure list. When there is only one pass over the document, WATCOM Script/GML will create the figure list at the end of the document.

9.40 FIGREF

```
Format: :FIGREF refid='id-name'
                [page=yes
                      no].
```

This tag causes a figure reference to be generated. The text "Figure" followed by the figure number will be generated at the point where the **:figref** tag is specified. The figure reference tag is a paragraph element, and is used with text to create the content of a basic document element. The figure being referenced must have a figure caption specified.

The **refid** attribute will determine the figure for which the reference will be generated. The specified identifier name must be the value of the id attribute on the figure you wish to reference.

The **page** attribute controls the output of the figure page number. If the attribute value *yes* is specified, the text "on page" followed by the page number of the referenced figure is placed after the figure reference text. If the attribute value *no* is specified, the page number of the referenced figure is not generated. If the page attribute is not specified, the figure page number is generated when the figure and the reference to it are not on the same output page.

9.41 FN

```
Format: :FN [id='id-name'].
            <paragraph elements>
            <basic document elements>
```

The footnote tag causes a note to be placed at the bottom of the page. The footnote text is preceded by a footnote number which is generated by the WATCOM Script/GML processor. Footnotes may be used where a basic document element is permitted to appear, with the exception of a figure, footnote, or example. The **:efn** tag terminates a footnote.

The **id** attribute assigns an identifier name to the footnote. The identifier name is used when processing a footnote reference, and must be unique within the document.

9.42 FNREF

```
Format: :FNREF refid='id-name'.
```

This tag causes a footnote reference to be generated. The number of the referenced footnote will be generated at the point where the **:fnref** tag is specified. The footnote reference tag is a paragraph element, and is used with text to create the content of a basic document element.

The **refid** attribute will determine the footnote for which the reference will be generated. The identifier name must be specified as the value for the id attribute on the footnote you wish to reference.

9.43 FRONTM

```
Format: :FRONTM.
```

This tag signals the start of the front material of a GML document, and must be preceded by the **:gdoc** tag.

9.44 GDOC

```
Format: :GDOC [sec='character string'].
```

This tag signals the start of a GML document and must precede all other document tags. All layout tags must precede the **:gdoc** tag.

The **sec** attribute will assign a security classification to the document. The attribute value may be used in the creation of banners which appear at the top and/or bottom of an output page.

9.45 GL

```
Format: :GL [compact]
            [termhi=term-highlight].
```

The glossary list tag signals the start of a glossary list, and is usually used in the back material section. Each list item in a glossary list has two parts. The first part is the glossary term and is defined with the **:gt** tag. The second part is the glossary description and is defined with the **:gd** tag. A corresponding **:egl** tag must be specified for each **:gl** tag.

The **compact** attribute indicates that the list items should be compacted. Blank lines that are normally placed between the list items will be suppressed. The compact attribute is one of the few WATCOM Script/GML attributes which does not have an attribute value associated with it.

The **termhi** attribute allows you to set the highlighting level of the glossary term. Non-negative integer numbers are valid highlighting values.

9.46 GD

```
Format: :GD.<paragraph elements>
            <basic document elements>
```

The glossary description tag signals the start of the text for an item in a glossary list. The glossary description tag must be preceded by a corresponding **:gt** tag, and may only appear in a glossary list.

9.47 GRAPHIC

```
Format: :GRAPHIC file='file name'
                 [depth='vert-space-unit']
                 [width=page
                        column
                        'hor-space-unit']
                 [scale=number]
                 [xoff='hor-space-unit']
                 [yoff='vert-space-unit'].
```

This tag is used to include a graphic image file into the document. WATCOM Script/GML supports two types of graphic include files. If the first two characters in the file are percent(%) followed by an exclamation mark(!), then the file is a PostScript graphic. A PostScript graphic file will only produce an image if the document is produced for a PostScript device. If the image file is not a PostScript graphic, a special validity check is performed on the file to determine if it is a WATCOM GKS PXA

image file. If it is not a PXA file, it is assumed to be a PostScript graphic file. PXA files are supported with PostScript, HP LaserJet Plus, and IBM PC Graphic printers, although grey scales are only supported with a PostScript device. Documents can be proofed on devices which are not supported by the graphic tag. If the device is not supported, the appropriate amount of white space is left for the graphic. All space value attributes are linked to the current font being used in the document (see "Font Linkage" on page 77).

The required attribute **file** specifies the name of the graphic file to include. The value of the attribute is a character string, and may be any valid file name.

The **depth** attribute specifies the vertical size of the graphic image. The value of the attribute is any valid vertical space unit, and must be specified if the graphic is a PostScript image. If the specified depth is less than the size of the actual graphic, the difference in size is taken off the top of the graphic image. If the depth is not specified when including a PXA file, the graphic depth is obtained from information within the image file. A PXA file is assumed to be defined in a vertical direction with 150 dots per inch (dpi) for PostScript and HP LaserJet devices, and 72dpi for PC Graphics printers.

The **width** attribute allows you to specify the width of the graphic. The attribute value *page* specifies that the graphic will be as wide as the page, even if the document is formatted for more than one column. The attribute value *column* specifies that the graphic shall be one column wide. If a horizontal space unit is used as the attribute value, the graphic will have the width specified by the attribute value. If the graphic is larger than the specified width, the difference in size is taken off the right hand side of the graphic image. A PXA file is assumed to be defined in a horizontal direction with 150 dots per inch (dpi) for PostScript and HP LaserJet devices, and 120dpi for PC Graphics printers.

The **scale** attribute allows you to alter the size of the graphic. The scale operation is performed after all depth and offset calculations are completed, and is supported with PostScript and HP LaserJet devices only. The attribute value is a positive integer number which represents a percentage of the original graphic size. Therefore, the value '100' will result in no scaling. With the HP LaserJet, only the values *50, 100, 150* and *200* are valid.

The **xoff** and **yoff** attributes specify an offset into the graphic. Some images are saved so that they will print in the middle of a blank page. By specifying the amount of space from the lower left corner of this blank page to the lower left hand corner of the printable graphic with the offset attributes, WATCOM Script/GML can shift the graphic to position it properly on the page. The value of the attributes can be a vertical space unit, with negative values being allowed.

9.48 GT

```
Format: :GT.<text line>
```

This tag is used to specify the term which is defined for each item in a glossary list. It is always followed by a **:gd** tag, which specifies the start of the text to define the term, and may only appear in a glossary list.

9.49 H0, H1, H2, H3, H4, H5, H6

```
Format: :Hn [id='id-name']
            [stitle='character string'].<text line>
        (n=0,1)
Format: :Hn [id='id-name'].<text line>
        (n=0,1,2,3,4,5,6)
```

These tags are used to create headings for sections and subsections of text. A common convention uses the headings as follows:

:H0 Major part of document.

:H1 Chapter.

:H2 Section.

:H3, *:H4*, *:H5*, *:H6* Subsections.

The specific layout with which a document is formatted will determine the format of the headings. Some layouts cause the headings to be automatically numbered according to a chosen convention. The heading text specified with the tag may also be used in the creation of top and/or bottom page banners.

A heading may be used where a basic document element is permitted to appear, with the following restrictions:

1. **:h0** tags may only be used in the body of a document.

2. **:h1** tags may not be used in the preface or the abstract.

The **stitle** attribute allows you to specify a short title for the heading. The short title will be used instead of the heading text when creating the top and/or bottom page banners. The short title attribute is valid with a level one or level zero heading.

The **id** attribute assigns an identifier name to the heading. The identifier name is used when processing a heading reference, and must be unique within the document.

9.50 HDREF

```
Format: :HDREF refid='id-name'
              [page=yes
                    no].
```

This tag causes a heading reference to be generated. The heading reference tag is a paragraph element, and is used with text to create the content of a basic document element. The heading text from the referenced heading is enclosed in double quotation marks and inserted into the formatted document.

The **refid** attribute will determine the heading for which the reference will be generated. The specified identifier name must be the value of the id attribute on the heading tag you wish to reference.

The **page** attribute controls the output of the heading page number. If the attribute value *yes* is specified, the text "on page" followed by the page number of the referenced heading is placed after the heading text. If the attribute value *no* is specified, the page number of the referenced heading is not generated. If the page attribute is not specified, the heading page number is generated when the heading and the reference to it are not on the same output page.

9.51 HP0, HP1, HP2, HP3

```
Format: :HPn.
        (n=0,1,2,3)
```

These tags start the highlighting of phrases at one of the four levels provided by GML. The actual highlighting to be performed is determined by the type of device for which the document is being formatted. Examples of highlighting include underlining, displaying in bold face, or using a different character shape (such as italics).

Highlighting may not be used when the GML layout explicitly determines the emphasis to be used, such as in the text of a heading.

The highlighting tags are paragraph elements. They are used with text to create the content of a basic document element, such as a paragraph. A corresponding **:EHPn** tag must be specified for each **:HPn** tag.

9.52 I1, I2, I3

```
Format: :In [id='id-name']
            [pg=start
                end
                major
                'character string']
            [refid='id-name'].<text line>
        (n=1,2,3)
```

These tags will cause an index entry to be created. Index entry tags may be used at any point in the document after the **:gdoc** tag. The text line with the index entry tag is used to create an index term for the index entry. The index command line option must be specified for the index entry tags to be processed. The **:I1** tag is used to create a primary index entry. The **:I2** tag is used to create an index subentry for the previous primary index entry. The **:I3** creates an index subentry for the previously specified **:I2** tag.

The **id** attribute assigns an identifier name to the created index entry. The identifier name is used by other tags when processing an index reference, and must be unique within the document.

The **pg** attribute determines the way in which the page number for the index entry is presented. If the attribute value is *start*, the index entry will have a page range. The *end* attribute value on an index entry will mark the end of a previously started page range. The attribute value *major* makes the page number reference of higher priority than the other page references in the index entry, and causes it to be listed first. If a character string is specified as the attribute value, the character string is placed in the index instead of a page number.

The **refid** attribute will cause the index entry to be associated with a specific higher level index entry rather than the index entry which directly precedes it in the document. The refid attribute may be used with the **:I2** and the **:I3** tags.

9.53 IH1, IH2, IH3

```
Format: :IHn [id='id-name']
             [ix=x]
             [print='character string']
             [see='character string']
             [seeid='id-name'].<text line>
        (n=1,2,3)
        (x=0 -> 8)
```

The index heading tags will cause an index entry to be created. Index headings may be used at any point in the document. The text line with the index entry tag is used to create an index term for the index entry. The index heading tag does not generate a page number reference with the index term in the index. The index command line option must be specified for the index entry tags to be processed. The **:IH1** tag is used to create a primary index entry. The **:IH2** tag is used to create an index subentry for the previous primary index entry. The **:IH3** creates an index subentry for the previously specified **:IH2** tag.

The **id** attribute assigns an identifier name to the created index entry. The identifier name is used by other tags when processing an index reference, and must be unique within the document.

The **ix** attribute selects one of the index groups (from zero through eight), with zero being the default.

The **print** attribute causes the specified character string to be displayed in the index instead of the index term. The index term is still used to determine where in the index the entry should be placed.

The **see** attribute will cause the supplied character string to be used as a page number reference. The character text "See" will prefix the character string in the index if there are no references in the index entry. If there are index subentries or page references, the string "See also" will be prefixed to the character string. It is your responsibility to ensure that index entries specified in the character string are actually in the index. The **see** attribute may only be used when the index entry is of level one or two.

The **seeid** attribute is used to reference an index entry. The index term created by the referenced index entry is used instead of a page number. If the referenced index entry has the **print** attribute specified, the character string value of the **print** attribute will be used instead of the index term. The character string "See" will prefix the index term in the index if there are no page references in the index entry. If there are index subentries or page references, the string "See also" will be prefixed to the index term. The **seeid** attribute may only be used when the index entry is of level one or two.

9.54 IMBED

```
Format: :IMBED file='file name'.
```

The value of the required attribute **file** is used as the name of the file to imbed. This tag is equivalent to the **:include** tag.

9.55 INCLUDE

```
Format: :INCLUDE file='file name'.
```

The value of the required attribute **file** is used as the name of the file to include. The content of the included file is processed by WATCOM Script/GML as if the data was in the original file. This tag provides the means whereby a document may be specified using a collection of separate files. Entering the source text into separate files, such as one file for each chapter, may help in managing the document.

If the specified file does not have a file type, the default document file type is used. For example, if the main document file is `manual.doc`, `doc` is the default document file type. If the file is not found, the alternate extension supplied on the command line is used. If the file is still not found, the file type `GML` is used.

When working on a PC/DOS system, the DOS environment symbol **GMLINC** may be set with an include file list. This symbol is defined in the same way as a library definition list (see "Defining a Library List" on page 297), and provides a list of alternate directories for file inclusion. If an included file is not defined in the current directory, the directories specified by the include path list are searched for the file. If the file is still not found, the directories specified by the DOS environment symbol **PATH** are searched.

9.56 INDEX

```
Format: :INDEX  [ix=n].
        (n=0 -> 8)
```

This tag may be used in the back material of a GML document to create the formatted index.

The **ix** attribute selects one of the index groups (from zero through eight), with zero being the default.

The index command line option must be specified for the index tag to be processed. All index tags are ignored if the option is not specified, allowing for faster draft document creation.

9.57 IREF

```
Format: :IREF refid='id-name'
              [pg=start
                  end
                  major
                  'character string']
              [see='character string']
              [seeid='id-name'].
```

This tag will cause an index entry to be created. The entry will be similar to the one referenced by the **refid** attribute, which must be specified. Index references may be placed anywhere in the document. The index command line option must be specified for the index reference tag to be processed.

The **refid** attribute is used to reference an index entry identified by the specified identifier name.

The **pg** attribute determines the way in which the page number for the index entry is presented. If the attribute value is *start*, the index entry will have a page range. The *end* attribute value on an index entry will mark the end of a previously started page range. The attribute value *major* makes the page number reference of higher priority than the other page references in the index entry, and causes it to be listed first. If a character string is specified as the attribute value, the character string is placed in the index instead of a page number.

The **see** attribute will cause the supplied character string to be used as a page number reference. The character text "See" will prefix the character string in the index if there are no references in the index entry. If there are index subentries or page references, the string "See also" will be prefixed to the character string. It is your responsibility to ensure that index entries specified in the character string are actually in the index. The **see** attribute may only be used when the index entry is of level one or two.

The **seeid** attribute is used to reference an index entry. The index term created by the referenced index entry is used instead of a page number. If the referenced index entry has the **print** attribute specified, the character string value of the **print** attribute will be used instead of the index term. The character string "See" will prefix the index term in the index if there are no page references in the index entry. If there are index subentries or page references, the string "See also" will be prefixed to the index term. The **seeid** attribute may only be used when the index entry is of level one or two.

9.58 LAYOUT

```
Format: :LAYOUT.
```

This tag starts a layout section. The layout tag is a special WATCOM Script/GML tag used to modify the default layout of the output document. More than one layout section may be present, but all layout sections must appear before the **:gdoc** tag. The **:elayout** tag terminates a layout section. See "Layouts" on page 121 for more information on the layout tag.

9.59 LI

```
Format: :LI [id='id-name'].<paragraph elements>
                           <basic document elements>
```

This tag signals the start of an item in a simple, ordered, or unordered list. The unordered list items are preceded by an annotation symbol, such as an asterisk. The ordered list items are annotated by an ordered sequence.

The **id** attribute associates an identifier name with the list item, and may only be used when the list item is in an ordered list. The identifier name is used when processing a list item reference, and must be unique within the document.

9.60 LIREF

```
Format: :LIREF refid='id-name'
               [page=yes
                     no].
```

This tag generates a reference to an item in an ordered list. The list item reference tag is a paragraph element, and is used with text to create the content of a basic document element. The number text from the referenced list item is inserted into the output.

The **refid** attribute will determine the list item for which the reference will be generated. The specified identifier name must be the value of the id attribute on the list item tag you wish to reference.

The **page** attribute controls the output of the list item page number. If the attribute value *yes* is specified, the text "on page" followed by the page number of the referenced list item is placed after the annotation text. If the attribute value *no* is specified, the page number of the referenced list item is not generated. If the page attribute is not specified, the list item page number is generated only if the list item reference is not on the same page as the list item.

9.61 LP

```
Format: :LP.<paragraph elements>
```

The list part tag is used to insert an explanation into the middle of a list. It may be used in simple, ordered, unordered, definition and glossary lists.

9.62 LQ

```
Format: :LQ.<basic document elements>
```

This tag starts a long quotation. WATCOM Script/GML does not surround a long quotation with quotes. The long quote is made distinct from the rest of the text by the way in which it is formatted. The **:elq** tag terminates a long quotation.

9.63 NOTE

```
Format: :NOTE.<paragraph elements>
```

This tag signals the start of a note. The paragraph elements are formatted with some emphasizing text, such as the default text "Note: ", in front of the paragraph elements.

A note may be used wherever a basic document element is permitted to appear.

9.64 OL

```
Format: :OL [compact].
```

This tag signals the start of an ordered list. Items in the list are specified using the **:li** tag. The list items are preceded by the number of the list item. The layout determines the style of the number.

An ordered list may be used wherever a basic document element is permitted to appear. A corresponding **:eol** tag must be specified for each **:ol** tag.

The **compact** attribute indicates that the list items should be compacted. Blank lines that are normally placed between the list items will be suppressed. The compact attribute is one of the few WATCOM Script/GML attributes which does not have an attribute value associated with it.

9.65 P

```
Format: :P.<paragraph elements>
```

This tag signals the start of a paragraph. Many layouts cause the first line of a paragraph to be indented.

A paragraph may occur wherever a basic document element is permitted.

9.66 PC

```
Format: :PC.<paragraph elements>
```

The paragraph continuation tag signals the start of a paragraph continuation. A paragraph continuation tag will be necessary when another basic document element, such as an example, is placed in the middle of a paragraph. Most layouts do not indent the first line of a paragraph continuation.

The tag may be used wherever a basic document element is permitted.

9.67 PREFACE

```
Format: :PREFACE.
```

This tag signals the start of the preface in the front material of a GML document. The preface is optional, but if specified it must follow the abstract and precede the table of contents. Basic document elements and heading levels two through six (:h2-:h6) may be specified in the preface.

9.68 PSC

```
Format: :PSC [proc='character string'].
```

This tag allows you to specify process-specific controls in your document. The **:psc** tag may be used anywhere in the document, and is terminated by the **:epsc** tag.

The **proc** attribute determines when the text in the psc block will be processed. If the proc attribute is not specified, the text in the psc block will always be processed. When the proc attribute is specified, the attribute value is a character string composed of device names separated by blanks. If the device being used to format the document matches one of the specified names in the list, the process control block is processed. In

addition to the device names, one other process name may be specified in the proc list. This name is checked against the name set by the **process** command line option.

9.69 Q

```
Format: :Q.
```

This tag starts a quote. The quote is enclosed in double quotation marks. When quotes are specified within other quotes, they are alternately enclosed by single and double quotation marks.

The quote tag is a paragraph element. It is used with text to create the content of a basic document element, such as a paragraph. A corresponding **:eq** tag must be specified for each **:q** tag.

9.70 SET

```
Format: :SET symbol='symbol-name'
             value='character-string'
                    delete.
```

This tag defines and assigns a value to a symbol name.

The **symbol** attribute must be specified. The value of this attribute is the name of the symbol being defined, and cannot have a length greater than ten characters. The symbol name may only contain letters, numbers, and the characters @, #, $ and underscore(_).

The **value** attribute must be specified. The attribute value *delete* or a valid character string may be assigned to the symbol name. If the attribute value *delete* is used, the symbol referred to by the symbol name is deleted. Refer to "Symbolic Substitution" on page 78 for more information about symbol substitution.

9.71 SF

```
Format: :SF font=number.
```

The set font tag starts the highlighting of phrases at the level specified by the required attribute *font*. The actual highlighting to be performed is determined by the type of device for which the document is being formatted. Examples of highlighting include underlining, displaying in bold face, or using a different character shape (such as italics).

The value of the *font* attribute is a non-negative integer number. If the specified number is larger than the last defined font for the document, font for zero is used.

Highlighting may not be used when the GML layout explicitly determines the emphasis to be used, such as in the text of a heading.

The set font tag is a paragraph element. It is used with text to create the content of a basic document element, such as a paragraph. A corresponding **:ESF** tag must be specified for each **:SF** tag.

9.72 SL

```
Format: :SL [compact].
```

This tag signals the start of a simple list. Items in the list are specified using the **:li** tag.

A simple list may occur wherever a basic document element is permitted to appear. A corresponding **:esl** tag must be specified for each **:sl** tag.

The **compact** attribute indicates that the list items should be compacted. Blank lines that are normally placed between the list items will be suppressed. The compact attribute is one of the few WATCOM Script/GML attributes which does not have an attribute value associated with it.

9.73 TITLE

```
Format: :TITLE [stitle='character string'].<text>
```

This tag is used to specify the title of the document. It may only appear in the front material title page. The **:title** tag is specified for each line of a multiple line title. The title text specified with the tag may also be used in the creation of top and/or bottom page banners. When more than one title line is specified, the first one is used in banner creation.

The **stitle** attribute allows you to specify a short title. When a short title is specified, it may be used instead of the title text when creating the top and/or bottom page banners.

9.74 TITLEP

```
Format: :TITLEP.
```

This tag signals the start of the title page of a GML document. It may only appear in the front material of a document. A corresponding **:etitlep** tag must be specified for the **:titlep** tag.

9.75 TOC

```
Format: :TOC.
```

This tag may be used in the front material of a GML document to create a formatted table of contents. More than one pass will be required to properly place the table of contents. When there is only one pass over the document, WATCOM Script/GML will create the table of contents at the end of the document.

9.76 UL

```
Format: :UL [compact].
```

This tag signals the start of an unordered list. Items in the list are specified using the **:li** tag. The list items are preceded by a symbol such as an asterisk or a bullet.

This tag may be used wherever a basic document element is permitted to appear. A corresponding **:eul** tag must be specified for each **:ul** tag.

The **compact** attribute indicates that the list items should be compacted. Blank lines that are normally placed between the list items will be suppressed. The compact attribute is one of the few WATCOM Script/GML attributes which does not have an attribute value associated with it.

9.77 XMP

```
Format: :XMP [depth='vert-space-unit'].
             <paragraph elements>
             <basic document elements>
```

This tag signals the start of an example. Each line of source text following the example tag is placed in the output document without normal text processing. Spacing between words is preserved, and the input text is not right justified. Input source lines which do not fit on a line in the output document are split into two lines on a character, rather than a word basis. An example may be used where a basic document element is permitted to appear, except within a figure, footnote, or example. A corresponding **:exmp** tag must be specified for each **:xmp** tag.

If the example does not fit on the current page or column, it is forced to the next one. If the current column is empty, the example will be split into two parts.

The **depth** attribute accepts vertical space units as possible values. The amount of specified vertical space is created in the output before any source input text is processed. The value of the depth attribute is linked to the current font (see "Font Linkage" on page 77).

10 GML Letter Tags

This section contains a subsection on each of the tags supported with the WATCOM Script/GML letter format. The tags are presented in alphabetical order, and are presented in the same format as the standard WATCOM Script/GML tags. Also note that the FORM command line option must be specified with the value LETTER (see "FORMat" on page 215).

10.1 ATTN

```
Format: :ATTN.attention name
```

The optional attention tag is used to identify a specific person or department at a general address. The default text "Attention: " followed by the attention name text is placed in the output document. The attention tag must be specified after the **:to** tag. The tags **:attn**, **:open**, and **:subject** may be specified in any order.

10.2 CLOSE

```
Format: :CLOSE [depth='vert-space-unit'].<text line>
         <author lines>
```

This tag closes the letter, and must be specified after the main body of the document. The close text line specifies the closing salutation. The closing salutation text is placed in the output document followed by a layout-determined delimiter (such as a comma). Each line following the close tag will be an output line of the author's signature and position. The **:eclose** tag will terminate the CLOSE.

The **depth** attribute accepts any valid vertical space unit. The specified amount of space is placed between the closing salutation and the author lines.

10.3 DATE

```
Format: :DATE [align=left
                      right]
              [depth='vert-space-unit'].<text line>
```

This tag specifies the date associated with the letter, and is specified after the **:from** tag. The current date is used if the optional date text line is not specified. The **:date** tag may be omitted from the letter.

The **align** attribute positions the date text on the output page. The attribute value *left* causes the date text to appear at the left margin of the letter. The attribute value *right* causes the date text to appear at the right margin of the letter. The value of the align attribute will be determined by the layout if it has not been specified.

The **depth** attribute accepts any valid vertical space unit. The specified amount of space is placed before the date text.

10.4 DIST

```
Format: :DIST.label
         <name lines>
```

The **:dist** tag starts a list of distribution destinations or enclosures, and is specified after the **:distrib** tag. The label text associated with the tag identifies the type of the distribution list. Each line following the dist tag is a distribution destination.

10.5 DISTRIB

```
Format: :DISTRIB.
```

The optional distribution tag starts a distribution or enclosure list after the close of the letter. The **:dist** tag is used to start each category in the distribution list.

10.6 DOCNUM

```
Format: :DOCNUM.document number
```

The optional document number tag specifies the number associated with the document, and is specified after the **:date** tag. The document number is not displayed on the letter page. It may be used in the banners at the top and/or bottom of the page.

10.7 ECLOSE

```
Format: :eCLOSE.typist mark
```

The **:eclose** tag is used to indicate the end of the close section. The optional text following the eclose tag is the typist mark, and is used to identify the person producing the letter.

10.8 EDISTRIB

```
Format: :eDISTRIB.
```

The **:edistrib** tag is used to indicate the end of the distribution section.

10.9 FROM

```
Format: :FROM.
         <address lines>
```

The **:from** tag starts an address entity. Each line following the tag will be a line in the address of the letter author. The first GML tag encountered will terminate the FROM address. If the paper on which the letter will be printed has the author's address on it, the **:from** tag may be omitted.

10.10 OPEN

```
Format: :OPEN.opening salutation
              <basic document elements>
```

The **:open** tag specifies the opening salutation text and must be specified. The salutation text is placed in the output document followed by a layout-determined delimiter (such as a colon). The body of the letter follows the open tag. The tags **:attn**, **:open**, and **:subject** may be specified in any order.

10.11 SUBJECT

```
Format: :SUBJECT.subject text
```

The optional subject tag is used to indicate the subject of the letter. The subject text is placed in the output document. The tags **:attn**, **:open**, and **:subject** may be specified in any order.

10.12 TO

```
Format: :TO [compact].
         <recipient lines>
```

The optional **:to** tag starts an address entity. Each line following the tag will be a line in the address of the letter recipient. The first GML tag encountered will terminate the TO address.

The optional attribute **compact** will suppress the printing of blank lines in the address. The option is most useful when printing form letters from a database which contains some empty fields.

11 Script Support

Script is a formatting language used at many installations for creating documents. The Script commands (control words) are format directives which define how a document is formatted. This is in contrast with the GML tags, which define the content of a document.

The Script directives are recognized and processed when the SCRIPT command line option is specified. Each document record which begins with a period in the first column is a Script control line. The period is called the *control word indicator*. A Script control line must contain a valid Script directive. All of the directives defined by the Waterloo Script product are recognized. Those control words not implemented are ignored. See "Processing Rules" on page 75 for details on the processing rules for a source document, and " UnProcessed Script Control Words" on page 301 for a list of Script control words which are not processed.

The control words have been implemented based on version 90.1 of the Waterloo Script product. The documentation for this product is included with the WATCOM Script/GML package to provide the documentation for the Script support. The control words for define macro (.DM), gml tag (.GT), and gml attribute (.GA) are also described in this document. These control words are the fundamental tools needed to build your own set of GML tags.

Many of the Script directives cause a break. A break will cause any text currently formatted on an output line to be sent to the output device. Any new text will not be joined with the previously processed document text.

11.1 Control Word Modifiers

Modifiers change the processing of the Script control line, and are placed immediately after the control word indicator. There are two modifiers for a control word specification.

The single quote(') modifier directs the processor to ignore control word separators(;) in the input line. The separator character will be treated as text data, and included in the processing of the control word operand.

If there are two control word indicators at the beginning of the logical record, the list of macros is not searched. The characters which follow must be a Script control word.

11.2 DM Control Word

```
  .DM  name  BEGIN
     macro data
  .DM  name END

or

  .DM  name  DELETE | OFF

or

  .DM  name  /data line1/data line2/.../data linen[/]
```

The **define macro** control word is used to create or remove a macro definition, and does not cause a break. Macros contain source fragments which may be processed by specifying the name of the macro. See "Processing Rules" on page 75 for details on the processing rules of a source document.

The first form of the .dm control word creates a macro. The name of a macro may be one to eight characters in length. All macro data lines are saved without processing (including symbol substitution) until the define macro END is recognized. The ending define macro control word must start at the beginning of the physical input line in the document source.

The DELETE macro option removes the specified macro name from the list of defined macros.

The last form of the .dm control word creates a macro from the operand line. The first character of the operand (in this case the / character) is used to delimit individual lines in the macro definition.

11.2.1 Invoking Macros

Macros are invoked in the document source by entering the control word indicator(.) in the first column of a logical record immediately followed by the name of the macro. All of the logical record text after the macro name is processed as parameter data for the macro. Invoking a macro does not cause a break.

Each operand value is separated by a space. Operand values may be enclosed in quotation marks if they contain a blank space. If a valid symbol name is immediately followed by an equals(=) sign, the value to the right hand side of the equals is assigned

to the symbol name. All other operands are assigned to symbols local to the macro. The symbol names used for these values are `&*1`, `&*2`, `&*3`, ..., `&*n` until all values have been assigned to a symbol. The symbol `&*0` contains the number of local symbols that have been created. The symbol `&*` contains the entire macro operand line.

If `screen` is the name of a defined macro, then it could be invoked as follows:

```
.screen file='example' '2.5i'
```

The first operand value defines the symbol `&file` with the value `example`. The second operand value assigns `2.5i` to the local macro symbol `&*1`. See "Symbolic Substitution" on page 78 for more information on symbols and symbol substitution.

11.3 GA Control Word

```
  .GA  tagname | *   attname | *   [options(A)]  [options(B)]

where options(A) are:

       OFF | ON
       UPpercase
       REQuired

where options(B) are:

       AUTOmatic   'string'
       LENgth      integer number
       RANge       minvalue   maxvalue   [default1  [default2]]
       VALue       'valname'  [USE       'string']  [DEFault]
       ANY         ['string']
       RESET       'valname' | 'string' | integer
```

The **GML attribute** control word defines or modifies an attribute for a GML tag. The *tagname* value must have been previously defined by a .GT control word. If an asterisk(*) is used, the last GML tag defined or operated on will be referenced. The *attname* value defines a new or modifies an existing tag attribute name. An attribute name must contain no more than nine alphanumeric characters. If an asterisk(*) is used, the last attribute name specified for the current tag will be referenced.

One or more of the option(A) values may be specified with the GML attribute control word.

OFF The attribute will be ignored when specified on a GML tag by the user.

ON Processing of an attribute which was previously ignored due to the OFF option is restarted.

UPpercase The value of the attribute is converted to uppercase before being processed.

REQuired The attribute must always be specified when the GML tag is used.

More than one option(B) value may be specified for a GML attribute, each of which must be specified by a separate .GA control word.

AUTOmatic The string value specified with this option is processed as the value of the attribute as if it was specified by the user. An automatic attribute may not be actually specified by the user with the GML tag.

LENgth The number specified with this option is the maximum number of characters accepted as an attribute value.

RANge The first two numbers specify the minimum and maximum numeric values allowed with the current attribute. The optional number *default1* provides a default value if the attribute is not specified with the tag. If the attribute is specified without a value, the optional number *default2* provides a default value (*default1* will be the default if *default2* is not specified).

VALue The option operand *valname* is defined as one of the possible values for an attribute. The VALUE option must be specified for each possible VALNAME you wish to define as a possible attribute value. If the `USE` keyword is specified, the USE string value is processed as the attribute value when the VALNAME value is specified. The `DEFault` keyword defines the default attribute value if the attribute is not specified with the GML tag.

ANY Any character string may be specified as the attribute value. If the optional string operand is also specified, it is used as the default value if the attribute is not specified with the GML tag.

RESET The reset option resets the current attribute values. With an AUTOMATIC or ANY attribute, the default string operand is reset. With a RANGE attribute, two numbers may be specified to reset the two default range numbers. With a VALUE attribute, the option will reset the default value to the specified value name.

11.4 GT Control Word

```
  .GT  tagname       ADD     macro-name   [tag options]

or

  .GT  tagname       CHAnge  macro-name

or

  .GT  tagname | *   DELete | PRint

or

  .GT  tagname       OFF | ON

where tag options are:

        ATTributes
        CONTinue
        CSOFF
        NOCONTinue
        TAGnext
        TEXTDef         'string'
        TEXTError
        TEXTReqd
```

The **GML tag** control word defines or modifies a GML tag. The *tagname* value must have been previously defined by a .GT control word for all but the ADD operand, and may not contain more than fifteen alphanumeric characters.

ADD Specifies a new GML tag and assigns the macro 'macro-name' to process the tag information. The tag options 'continue', 'nocontinue', and 'tagnext' are recognized but not currently supported.

ATTributes The GML tag has one or more attributes.

CONTinue Each tag is treated as though it starts on a new input line. The 'continue' option causes a continue character to be generated before processing the tag.

CSOFF This option will terminate any active process control (or conditional) sections.

NOCONTinue The current tag cannot be continued by a previous tag.

TAGnext Document text is not allowed after the current tag. Another GML tag must follow in the input.

TEXTDef The specified character string is used if tag text is not specified with the tag.

TEXTError Tag text is not allowed with the tag.

TEXTLine All data to the end of the input line is treated as tag text.

TEXTReqd Tag text must be specified with the tag.

CHAnge The macro processor for the current GML tag is reassigned to be the macro 'macro-name'.

DELete The current GML tag and its associated attributes are deleted, and will no longer be recognized as a GML tag. If an asterisk(*) is specified as the tag name, all GML tags are deleted.

OFF The GML tag will be not be processed if found in the document.

ON Processing of a tag which was previously ignored due to the OFF option is restarted.

PRint The current GML tag and its associated attributes are printed on the output screen. If an asterisk(*) is specified as the tag name, all GML tags are printed.

12 Layouts

12.1 Specifying and Using Layouts

The layout determines the way in which the document elements specified by the GML tags are formatted on the output page. Many of the formatting actions may be modified through the supplied layout tags. The layout tags are specified in much the same way as the GML tags are specified in the document. Some of the layout tags, such as **:fig**, have the same name as the GML tags which they modify.

A layout section starts with the **:layout** tag, and must appear before the **:gdoc** tag. The layout section is terminated with an **:elayout** tag. Only the portions of the layout you wish to change need to be specified, as the changes modify the default layout which is built into WATCOM Script/GML. If more than one layout is specified, the changes are cumulative. With the exception of the **:banner** and **:banregion** tags, the attributes of the layout tags are all optional.

The layout section may be stored in a separate file. This file may be included at the start of the document with the **:include** tag, or specified when you run WATCOM Script/GML with the LAYOUT command line option. Including the layout with either of these two methods makes it easier to select a different layout.

The **:convert** tag may be used to determine the attribute values in the current layout.

12.2 Number Style

The term **number style** is used throughout the layout section of this document. The number style is a sequence of up to three codes which defines the style of a generated number. The first code indicates the form of the number digits.

A The number is formed with lower case alphabetic characters. Example: **28** is represented by **ab** while **29** is represented by **ac**. (a=1, b=2,..., z=26, aa=27, ab=28)

B The number is formed with upper case alphabetic characters. Example: **28** is represented by **AB** while **29** is represented by **AC**. (A=1, B=2,..., Z=26, AA=27, AB=28)

H The number is formed with hindu-arabic characters. Example: The number twenty eight is represented by **28**.

R The number is formed with lower case roman numerals. Example: The number **28** is represented by **xxviii**.

C The number is formed with upper case roman numerals. Example: The number **28** is represented by **XXVIII**.

The second code, which does not have to be specified, defines how the number is separated from other numbers or text.

D The number is followed by a decimal point.

P The number is surrounded by parentheses.

The third code may be specified if parentheses were specified in the second code.

A The number is preceded by a left parenthesis and is not followed by a right parenthesis.

B The number is followed by a right parenthesis and is not preceded by a left parenthesis.

12.3 Layout Tags

This section contains a subsection on each of the layout tags supported by WATCOM Script/GML. The tags are presented in alphabetical order, each with an example. Most of the example values are the values used with the default layout. The **:convert** tag can be used to determine the exact values.

12.3.1 ABSTRACT

Define the characteristics of the abstract section and the abstract heading.

```
:ABSTRACT
            post_skip = 1
            pre_top_skip = 1
            font = 1
            spacing = 1
            header = yes
            abstract_string = "ABSTRACT"
            page_eject = yes
            page_reset = yes
            columns = 1
```

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the abstract heading. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page. If the abstract heading is not displayed (the header attribute has a value of NO), the post-skip value has no effect.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the abstract heading. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the abstract heading. The font value is linked to the pre_top_skip and post_skip attributes (see "Font Linkage" on page 77).

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the abstract section.

header The header attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the abstract heading is generated. If the value *no* is specified, the header text is not generated.

abstract_string This attribute accepts a character string. If the abstract header is generated, the specified string is used for the heading text.

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the

heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

page_reset This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the page number is reset to one at the beginning of the section. With the :ABSTRACT tag only, a value of *yes* will cause the page number to always be reset after the title page.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the abstract.

12.3.2 ADDRESS

Define the characteristics of the address entity.

```
:ADDRESS
          left_adjust = 0
          right_adjust = '1i'
          page_position = right
          font = 0
          pre_skip = 2
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the address between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the address. The font value is linked to the left_adjust, right_adjust and pre_skip attributes of the :ADDRESS tag, and the skip attribute of the :ALINE tag (see "Font Linkage" on page 77).

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the address. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

12.3.3 ALINE

Define the characteristics of the address line entity.

```
:ALINE
        skip = 1
```

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between address lines.

12.3.4 APPENDIX

Define the characteristics of the appendix section and appendix heading. All of the attributes, with the exception of the spacing value, apply to the **:h1** tag while in the appendix section.

```
:APPENDIX
            indent = 0
            pre_top_skip = 0
            pre_skip = 0
            post_skip = 3
            spacing = 1
            font = 3
            number_font = 3
            number_form = new
            page_position = left
            number_style = b
            page_eject = yes
            line_break = yes
            display_heading = yes
            number_reset = yes
            case = mixed
            align = 0
            header = yes
            appendix_string = "APPENDIX "
            page_reset = no
            section_eject = yes
            columns = 1
```

indent The indent attribute accepts any valid horizontal space unit. The indent value is added to the offset determined by the page position attribute, giving the starting offset from the left margin for the appendix heading.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the appendix heading. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the appendix heading. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page. If the appendix heading is not displayed, the post-skip is ignored.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will

therefore be the spacing value minus one. The spacing attribute defines the line spacing within the appendix section.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the appendix heading. The font value is linked to the indent, post_skip and pre_top_skip attributes (see "Font Linkage" on page 77).

number_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The number font attribute defines the font of the appendix number.

number_form This attribute accepts the values *none*, *prop*, and *new*. The specified value determines the format of the appendix heading number. The value *none* indicates that no number is to be output. The value *prop* indicates that the number is composed of the number for the current level prefixed by the number for the previous level and the number delimiter specified in the **heading** layout tag. The value *new* indicates that only the number of the current level is to be output.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the appendix heading between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

number_style This attribute sets the number style of the appendix heading number. (See "Number Style" on page 121).

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

line_break This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the skip value specified by the post_skip attribute will be issued. If the value *no* is specified, the skip value specified by the post_skip attribute will be ignored. If a paragraph follows the heading, the paragraph text will start on the same line as the heading.

display_heading This attribute accepts the keyword values *yes* and *no*. The heading is not produced when the value *no* is specified. The heading pre and post skips are still generated.

number_reset This attribute accepts the keyword values *yes* and *no*. When a heading is processed, all heading levels after it have their heading numbers reset. When the value 'no' is specified, the number of the next level of heading is not reset.

case This attribute accepts the keyword values *mixed*, *upper* and *lower*. When a heading is processed, the text is converted to upper or lower case when the values UPPER or LOWER are used. The text is left unchanged when the value MIXED is used.

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the appendix heading. After the appendix heading is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the appendix heading.

header The header attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the appendix header (specified by the *appendix_string* attribute) is generated at the beginning of the heading text specified by a **:h1** tag. If the value *no* is specified, the header text is not generated.

appendix_string This attribute accepts a character string. If the appendix header is generated, the specified string is inserted before the **:h1** heading text.

page_reset This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the page number is reset to one at the beginning of the section.

section_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the section is not forced to a new page. The section is always placed on a new page when the value *yes* is specified.

The values *odd* and *even* will place the section on a new page if the parity (odd or even) of the current page number does not match the specified value.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the appendix.

12.3.5 ATTN

Define the characteristics of the attention entity in the letter format.

```
:ATTN
        left_adjust = 0
        page_position = left
        pre_top_skip = 1
        font = 1
        attn_string = "Attention: "
        string_font = 1
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the attention text between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the attention text. The pre-top-skip will be merged with the

previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the attention text. The font value is linked to the left_adjust and pre_top_skip attributes (see "Font Linkage" on page 77).

attn_string This attribute accepts a character string. The specified string precedes the attention text in the output document.

string_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The *string_font* attribute defines the font of the the attention string defined by the *attn_string* attribute.

12.3.6 AUTHOR

Define the characteristics of the author entity.

```
:AUTHOR
          left_adjust = 0
          right_adjust = '1i'
          page_position = right
          font = 0
          pre_skip = 25
          skip = 1
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the author line between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the

text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the author lines. The font value is linked to the left_adjust, right_adjust, pre_skip and skip attributes (see "Font Linkage" on page 77).

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the author lines. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between author lines.

12.3.7 BACKM

Define the characteristics of the back material section.

```
:BACKM
        post_skip = 0
        pre_top_skip = 0
        header = no
        backm_string = ""
        page_eject = yes
        page_reset = no
        columns = 1
        font = 1
```

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the back material. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of

an output page, any remaining part of the skip is not carried over to the next output page. If the back material heading is not displayed (the header attribute has a value of NO), the post-skip value has no effect.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the back material. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

header The header attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the back material heading is generated. If the value *no* is specified, the header text is not generated.

backm_string This attribute accepts a character string. If the back material header is generated, the specified string is used for the heading text.

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

page_reset This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the page number is reset to one at the beginning of the section.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the back material.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM

Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the header attribute value. The font value is linked to the pre_top_skip and post_skip attributes (see "Font Linkage" on page 77).

12.3.8 BANNER

Defines a page banner. A page banner appears at the top and/or bottom of a page. Information such as page numbers, running titles and the current heading would be defined in a banner. Banners may be defined for the top and/or bottom of a page in each section of the document. The banner attributes specify the size of the banner and the document section in which it is to be used.

A banner definition begins with the **:banner** tag and ends with the **:ebanner** tag. The banner is divided into a number of regions, each defined by the **:banregion** tag. The banner region definitions are placed after the banner attributes and before the **:ebanner** tag.

```
:BANNER
          left_adjust = 0
          right_adjust = 0
          depth = 3
          place = bottom
          refplace = bottom
          docsect = head0
          refdoc = body
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

depth The depth attribute accepts as its value any valid vertical space unit. It specifies the vertical depth of the banner.

place The place attribute specifies where on the odd or even numbered output page the banner is to be placed. The following values may be specified for this attribute:

topodd	The top of odd pages.
topeven	The top of even pages.
botodd	The bottom of odd pages.

boteven The bottom of even pages.

refplace The refplace attribute specifies the place value of an existing banner.

docsect The document section for which the banner will be used. The following values may be specified for this attribute:

abstract The banner will appear in the abstract section of the document.

appendix The banner will appear in the appendix section of the document.

backm The banner will appear in the back material section of the document.

body The banner will appear in the body section of the document.

figlist The banner will appear in the figure list section of the document.

HEADn The banner will appear when a heading of level **n**, where **n** may have a value of zero through six inclusive, appears on the output page.

letfirst The banner will appear on the first page of the letter when the letter format is used. If the letter has only one page, only the banner defined for the top of the page will be used. Even page banners are not allowed if *letfirst* is the document section value.

letlast The banner will appear on the last page of the letter when the letter format is used. If the letter has only one page, only the banner defined for the bottom of the page will be used.

letter The banner will appear on the pages between the first and last page of the letter when the letter format is used.

index The banner will appear in the index section of the document.

preface The banner will appear in the preface section of the document.

toc The banner will appear in the table of contents section of the document.

refdoc The refdoc attribute specifies the docsect value of an existing banner.

The *refplace* and *refdoc* attributes are used in combination to specify an existing banner. The referenced banner is copied to the banner being defined. These attributes are most commonly used when duplicating a banner for an odd or even page. When these attributes are specified, only the *place* and *docsect* attributes are required. All other attributes will override the attribute values of the banner being copied. If the two reference attributes are not specified, all of the other attributes are required.

To delete a banner, specify only the *place* and *docsect* attributes, and delete the individual banner regions.

12.3.9 BANREGION

Define a banner region within a banner. Each banner region specifies a rectangular section of the banner. A banner region begins with a **:banregion** tag and ends with an **:ebanregion** tag. All banner regions are defined after the banner tag attributes and before the **:ebanner** tag.

```
:BANREGION
             indent = 0
             hoffset = left
             width = extend
             voffset = 2
             depth = 1
             font = 0
             refnum = 1
             region_position = left
             pouring = last
             script_format = yes
             contents = '/&$htext0.// &$pgnuma./'
```

indent The indent attribute accepts any valid horizontal space unit. The specified space value is added to the value of the horizontal offset attribute (hoffset) to determine the start of banner region in the banner if the horizontal offset is specified as *left*, *centre*, or *center*. If the horizontal offset is specified as *right*, the indent value is subtracted from the right margin of the banner.

hoffset The hoffset attribute specifies the horizontal offset from the left side of the banner where the banner region will start. The attribute value may be any valid horizontal space unit, or one of the keywords *left*, *center*, *centre*, or *right*. The keyword values remove the dependence upon the left and right adjustment settings of the banner that occurs when using an absolute horizontal offset.

width This attribute may be any valid horizontal space unit, or the keyword *extend*. If the width of the banner region is specified as

extend, the width of the region will be increased until the start of another banner region or the right margin of the banner is reached.

voffset This attribute accepts any valid vertical space unit. It specifies the vertical offset from the top of the banner for the start of the banner region. A value of zero will be the first line of the banner, while the value one will be the second line of the banner.

depth The depth attribute accepts a vertical space unit value. The attribute value specifies the number of output lines or vertical space of the banner region.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the banner region's contents.

refnum This attribute accepts a positive integer number. Each banner region must have a unique reference number. If this is the only attribute specified, the banner region is deleted from the banner.

region_position This attribute specifies the position of the data within the banner region. The attribute value may be one of the keywords *left*, *center*, *centre*, or *right*.

pouring When the value of the contents attribute is a heading, and a heading of the specified level does not appear on the output page, the contents can be 'poured' back to a previous heading level. When the attribute value *none* is specified, no pouring occurs. In this case, the region will be empty. When the attribute value *last* is specified, the last heading appearing in the document with the same level as the heading specified by the contents attribute is used. The attribute value *headn*, where **n** may have a value of zero through six inclusive, may be specified. In this case, the last heading appearing in the document which has a level between zero and the pouring value is used.

script_format This attribute determines if the contents region is processed as a Script content string in the same way as the operand of a Script running title control word. If the attribute value is *yes*, then the value of the content attribute is treated as a Script format title string.

contents This attribute defines the content of the banner region. If the content value does not fit in the banner region, the value is truncated. Symbols containing the values for each of the content keywords are also listed. Specifying these symbols as part of the string content may be used to create more complex banner region values. Note that when using a symbol in a content string of a banner definition, you will need to protect it from being substituted during the definition with the &AMP symbol (ie &AMP.AUTHOR.). The possible values are:

author The first author of the document will be used. The symbol $AUTHOR is also defined with this value.

bothead The last heading on the output page is used. The symbol $BOTHEAD is also defined with this value.

date The current date will be used.

docnum The document number will be the content of the banner region. The symbol $DOCNUM is also defined with this value.

HEADn The last heading of level **n**, where **n** may have a value of zero through six inclusive. Both the heading number and heading text are both used. The symbols $HEAD0 through $HEAD6 are also defined with this value.

HEADNUMn The heading number from the last heading of level **n**, where **n** may have a value of zero through six inclusive. The symbols $HNUM0 through $HNUM6 are also defined with this value.

HEADTEXTn The text of the heading from the last heading of level **n**, where **n** may have a value of zero through six inclusive. If the *stitle* attribute was specified for the selected heading, the stitle value is used. The symbols $HTEXT0 through $HTEXT6 are also defined with this value.

none The banner region will be empty.

pgnuma The content of the banner region will be the page number of the output page in the hindu-arabic numbering style. The symbol $PGNUMA is also defined with this value.

pgnumad The content of the banner region will be the page number of the output page in the hindu-arabic numbering style followed by a decimal point. The symbol $PGNUMAD is also defined with this value.

pgnumr The content of the banner region will be the page number of the output page in the lower case roman numbering style. The symbol $PGNUMR is also defined with this value.

pgnumrd The content of the banner region will be the page number of the output page in the lower case roman numbering style followed by a decimal point. The symbol $PGNUMRD is also defined with this value.

pgnumc The content of the banner region will be the page number of the output page in the upper case roman numbering style. The symbol $PGNUMC is also defined with this value.

pgnumcd The content of the banner region will be the page number of the output page in the upper case roman numbering style followed by a decimal point. The symbol $PGNUMCD is also defined with this value.

rule The content of the banner region will be a rule line which fills the entire region.

sec The security value specified by the *sec* attribute on the **:gdoc** tag is used. The symbol $SEC is also defined with this value.

stitle The *stitle* attribute value from the first title tag specified in the front material of the document is used. If the stitle attribute was not specified, the title text is used. The symbol $STITLE is also defined with this value.

title The text of the first title tag specified in the front material of the document is used. The symbol $TITLE is also defined with this value.

string Any character string enclosed in quotation marks.

time The current time will be used.

tophead The first heading on the output page is used. The symbol $TOPHEAD is also defined with this value.

If a banner region does not already exist, then all attributes must be specified. If you wish to modify an existing banner region, the *refnum* attribute will uniquely identify the region. When the reference number is that of an existing banner region, all other attributes will modify the values of the existing banner region.

To delete a banner region, specify only the *refnum* attribute. All banner regions must be deleted before a banner definition will be removed.

12.3.10 BODY

Define the characteristics of the body section.

```
:BODY
        post_skip = 0
        pre_top_skip = 0
        header = no
        body_string = ""
        page_eject = yes
        page_reset = yes
        font = 1
```

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the body. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page. If the body heading is not displayed (the header attribute has a value of NO), the post-skip value has no effect.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied

by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the body. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

header The header attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the body heading is generated. If the value *no* is specified, the header text is not generated.

body_string This attribute accepts a character string. If the body header is generated, the specified string is used for the heading text.

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

page_reset This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the page number is reset to one at the beginning of the section.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the header attribute value. The font value is linked to the pre_top_skip and post_skip attributes (see "Font Linkage" on page 77).

12.3.11 CIT

Define the characteristics of the citation entity.

```
:CIT
       font = 1
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the citation text.

12.3.12 CLOSE

Define the characteristics of the close entity in the letter format.

```
:CLOSE
         pre_skip = 2
         depth = 6
         font = 0
         page_position = centre
         delim = ',’
         extract_threshold = 2
```

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the close. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

depth The depth attribute accepts as its value any valid vertical space unit. The value specifies the amount of space to be left after the text of the **:close** tag.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the closing statements. The font value is linked to the depth and pre_skip attributes (see "Font Linkage" on page 77).

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the closing statements between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the

text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

delim The delimiter attribute sets the delimiter character to be used following the closing salutation.

extract_threshold The depth attribute accepts as its value a positive integer number. If the text associated with the **:close** tag starts on a new page, the number of lines specified by the *extract_threshold* attribute will move to the next page with the closing text.

12.3.13 CONVERT

Convert the current layout into the specified file name. The resulting file will contain the entire layout in a readable form.

```
:CONVERT file='file name'.
```

12.3.14 DATE

Defines the characteristics of the date entity in the standard tag format. The **:letdate** layout tag defines the characteristics of the date entity in the letter tag format.

```
:DATE
        date_form = "$ml $dsn, $yl"
        left_adjust = 0
        right_adjust = '1i'
        page_position = right
        font = 0
        pre_skip = 2
```

date_form The date_form attribute accepts a character string value which defines the format of the date string. The year, month and day may be specified separately and in any order by special date sequences. These date sequences are started with a dollar($) sign and followed by one to three characters. Text which is not recognized as a date sequence can be entered to tailor the format of the resulting date.

The first character in a date sequence is a *Y* for the year, an *M* for the month, or a *D* for the day. The next character is the *L* or *S* character to specify the long or short form of the date sequence. If neither of these characters are present, the long form is used. When the length specifier is present, the *N* character is used to format the month or the day as a number. If the length specified is not present,

the month and day values are created in character form. The year is always formatted as a number.

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the date between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the date text. The font value is linked to the left_adjust, right_adjust and pre_skip attributes (see "Font Linkage" on page 77).

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the date. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

12.3.15 DD

Define the characteristics of the data description entity.

```
:DD
      line_left = '0.5i'
      font = 0
```

line_left This attribute accepts any valid horizontal space unit. The specified amount of space must be available on the output line after the definition term which precedes the data description. If there is not

enough space, the data description will be started on the next output line.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the data description.

12.3.16 DDHD

Define the characteristics of the data description heading entity.

```
:DDHD
        font = 1
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the data description heading.

12.3.17 DEFAULT

Define default characteristics for document processing.

```
:DEFAULT
           spacing = 1
           columns = 1
           font = 0
           justify = yes
           input_esc = ' '
           gutter = '0.5i'
           binding = 0
```

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the document when there is no layout entry for spacing with a specific document element.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created on each output page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the document when the font is not explicitly determined by the document element.

justify The justify attribute accepts the keyword values *yes* and *no*. Right justification of text is performed if this attribute has a value of *yes*. If justification is not desired, the value should be *no*.

input_esc The input escape attribute accepts the keyword value *none* or a quoted character. Input escapes are not recognized if the attribute value is *none* or a blank. If a character is specified as the attribute value, this character is used as the input escape delimiter. If an empty('') or *none* value is specified, the blank value is used. Refer to "Input Translation" on page 80 for more information.

gutter The gutter attribute specifies the amount of space between columns in a multi-column document, and has no effect in a single column document. This attribute accepts any valid horizontal space unit.

binding The binding attribute accepts any valid horizontal space unit. The binding value is added to the current left and right margins of those output pages which are odd numbered.

12.3.18 DISTRIB

Define the characteristics of the distribution list entity.

```
:DISTRIB
          pre_top_skip = 3
          skip = 1
          font = 0
          indent = '0.5i'
          page_eject = no
```

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before

the distribution list. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between items of the distribution list.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the items of the distribution list. The font value is linked to the indent, skip and pre_top_skip attributes (see "Font Linkage" on page 77).

indent The indent attribute accepts any valid horizontal space unit. The indent value is the offset from the left margin for the distribution list.

page_eject This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the distribution list is placed on a new page.

12.3.19 DOCNUM

Define the characteristics of the document number entity.

```
:DOCNUM
          left_adjust = 0
          right_adjust = '1i'
          page_position = right
          font = 0
          pre_skip = 2
          docnum_string = "Document Number "
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the document number between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the text specified by the *document_string* attribute and the document number. The font value is linked to the left_adjust, right_adjust and pre_skip attributes (see "Font Linkage" on page 77).

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the document number. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

docnum_string This attribute accepts a character string. The specified string precedes the document number in the output document.

12.3.20 DL

Define the characteristics of the definition list entity.

```
:DL
      level = 1
      left_indent = 0
      right_indent = 0
      pre_skip = 1
      skip = 1
      spacing = 1
      post_skip = 1
      align = '1i'
      line_break = no
```

level This attribute accepts a positive integer number. If not specified, a level value of '1'. is assumed. Each list level is separately specified. For example, if two levels of the ordered list are specified, the **:dl** tag will be specified twice in the layout. When some attributes for a new level of a list are not specified, the default

values for those attributes will be the values of the first level. Since list levels may not be skipped, each new level of list must be sequentially defined from the last specified level.

If there is an ordered, simple, and second ordered list nested together in the document, the simple and first ordered list will both be from level one, while the last ordered list will be level two. The appropriate level number is selected based on the nesting level of a particular list type. If a list type is nested beyond the levels specified in the layout, the levels are "cycled". For example, if there are two levels of ordered list specified in the layout, and there are three ordered lists nested, the third level of ordered list will use the attributes of the level one ordered list. A fourth nested list would use the attributes of the level two.

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the definition list.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the definition list.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the definition list. The pre-skip will be merged with the previous document entity’s post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between each item of the definition list.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the items of the definition list.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the definition list. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the definition term. After the definition term is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the definition list item.

line_break This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the data description starts a new line after the definition term if the length of the term is larger than align value. If the value *no* is specified, the definition term is allowed to intrude into the data description area.

12.3.21 DT

Define the characteristics of the definition term entity.

```
:DT
      font = 2
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the definition term. The font value is linked to the left_indent, right_indent, pre_skip, post_skip, skip and align attributes of the **:dl** tag, and the line_left attribute of the **:DD** tag (see "Font Linkage" on page 77).

12.3.22 DTHD

Define the characteristics of the definition term heading entity.

```
:DTHD
        font = 1
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the definition term heading.

12.3.23 EBANNER

Mark the end of a banner definition.

```
:eBANNER
```

12.3.24 EBANREGION

Mark the end of a banner region definition.

```
:eBANREGION
```

12.3.25 ECLOSE

Mark the end of the close entity in the letter tag format.

```
:eCLOSE
          pre_skip = 1
          font = 0
```

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the typist mark. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the typist mark. The font value is linked to the pre_skip attribute (see "Font Linkage" on page 77).

12.3.26 ELAYOUT

Mark the end of a layout definition.

```
:eLAYOUT
```

12.3.27 FIG

Define the characteristics of the figure entity.

```
:FIG
       left_adjust = 0
       right_adjust = 0
       pre_skip = 2
       post_skip = 0
       spacing = 1
       font = 0
       default_place = top
       default_frame = rule
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the figure. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the figure. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page. figure.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the figure.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the figure text. The font value is linked to the left_adjust, right_adjust, pre_skip and post_skip attributes (see "Font Linkage" on page 77).

default_place This attribute accepts the values *top*, *bottom*, and *inline*. The specified attribute value is used as the default value for the place attribute of the GML figure tag.

default_frame This attribute accepts the values *rule*, *box*, *none*, and *'character string'*. The specified attribute value is used as the default value for the frame attribute of the GML figure tag. See the discussion about the frame attribute under "FIG" on page 92 for an explanation of the attribute values.

12.3.28 FIGCAP

Define the characteristics of the figure caption entity.

```
:FIGCAP
          pre_lines = 1
          font = 0
          figcap_string = "Figure "
          string_font = 0
          delim = '.'
```

pre_lines This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting number of lines are skipped before the figure caption. If the document entity starts a new page, the specified number of lines are still skipped. The pre-lines value is not merged with the previous document entity's post-skip value.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the figure caption text. The font value is linked to the pre_lines attribute (see "Font Linkage" on page 77).

figcap_string This attribute accepts a character string. The specified string is the first part of the figure caption generated by WATCOM Script/GML.

string_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The *string_font* attribute defines the font of the the figure caption string defined from the text specified by the *figcap_string* attribute to the figure caption delimiter inclusive.

delim This attribute accepts a quoted character value. The delimiter value specifies the character which is inserted after the number of the figure. If a character other than a blank space is specified, that character followed by a blank space will be inserted. If a blank space is specified, only that blank space will be inserted.

12.3.29 FIGDESC

Define the characteristics of the figure description entity.

```
:FIGDESC
          pre_lines = 1
          font = 0
```

pre_lines This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting number of lines are skipped before the figure description. If the document entity starts a new page, the specified number of lines are still skipped. The pre-lines value is not merged with the previous document entity's post-skip value. If the previous tag was **:figcap**, this value is ignored.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM

Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the figure description. The font value is linked to the pre_lines attribute (see "Font Linkage" on page 77).

12.3.30 FIGLIST

Define the characteristics of the figure list.

```
:FIGLIST
          left_adjust = 0
          right_adjust = 0
          skip = 0
          spacing = 1
          columns = 1
          fill_string = "."
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between figure list items.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the figure list.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the figure list.

fill_string This attribute accepts a string value which is used to 'fill' the line between the text and the page number.

12.3.31 FLPGNUM

Define the characteristics of the figure list page numbers.

```
:FLPGNUM
          size = '0.4i'
          font = 0
```

size This attribute accepts any valid horizontal space unit. The specified value is the minimum amount of space that will be reserved on the output line for the figure page number.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the page number. The font value is linked to the size attribute (see "Font Linkage" on page 77).

12.3.32 FN

Define the characteristics of the footnote entity.

```
:FN
      line_indent = 0
      align = '0.4i'
      pre_lines = 2
      skip = 2
      spacing = 1
      font = 0
      number_font = 0
      number_style = h
      frame = none
```

line_indent The line_indent attribute accepts any valid horizontal space unit. This attribute specifies the amount of indentation for the first output line of the footnote.

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the footnote number. After the footnote number is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the footnote.

pre_lines This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied

by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting number of lines are skipped before the footnotes are output. If the document entity starts a new page, the specified number of lines are still skipped. The pre-lines value is not merged with the previous document entity's post-skip value.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between the footnotes.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the footnote.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the footnote text. The font value is linked to the line_indent, pre_lines, skip and align attributes (see "Font Linkage" on page 77).

number_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The number font attribute defines the font of the footnote number.

number_style This attribute sets the number style of the footnote number. (See "Number Style" on page 121).

frame This attribute accepts the values *rule* or *none*. If the value *rule* is specified, a rule line is placed between the main body of text and the footnotes at the bottom of the output page. If the footnote is placed across the entire page, the width of the rule line is half the width of the page. If the footnote is one column wide, the rule line width is the width of a column minus twenty percent.

12.3.33 FNREF

Define the characteristics of the footnote reference entity.

```
:FNREF
        font = 0
        number_style = hp
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the footnote reference text.

number_style This attribute sets the number style of the footnote reference number. (See "Number Style" on page 121).

12.3.34 FROM

Define the characteristics of the FROM entity in the letter tag format.

```
:FROM
        left_adjust = 0
        page_position = right
        pre_top_skip = 6
        font = 0
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the from text between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the FROM text. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the FROM text. The font value is linked to the left_adjust, page_position and pre_top_skip attributes (see "Font Linkage" on page 77).

12.3.35 GD

Define the characteristics of the glossary description entity.

```
:GD
      font = 0
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the glossary description.

12.3.36 GL

Define the characteristics of the glossary list entity.

```
:GL
      level = 1
      left_indent = 0
      right_indent = 0
      pre_skip = 1
      skip = 1
      spacing = 1
      post_skip = 1
      align = 0
      delim = ':'
```

level This attribute accepts a positive integer number. If not specified, a level value of '1'. is assumed. Each list level is separately specified. For example, if two levels of the ordered list are specified, the **:gl** tag will be specified twice in the layout. When some attributes for a new level of a list are not specified, the default values for those attributes will be the values of the first level. Since list levels may not be skipped, each new level of list must be sequentially defined from the last specified level.

If there is an ordered, simple, and second ordered list nested together in the document, the simple and first ordered list will both be from level one, while the last ordered list will be level two. The appropriate level number is selected based on the nesting level of a particular list type. If a list type is nested beyond the levels specified in the layout, the levels are "cycled". For example, if there are two levels of ordered list specified in the layout, and there are three ordered lists nested, the third level of ordered list will use the attributes of the level one ordered list. A fourth nested list would use the attributes of the level two.

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the glossary list.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the glossary list.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the glossary list. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between each item of the glossary list.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the items of the glossary list.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after

the glossary list. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the glossary term. After the glossary term is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the glossary list item.

delim The quoted character value is used to separate the glossary term from the glossary description.

12.3.37 GT

Define the characteristics of the glossary term entity.

```
:GT
      font = 2
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the glossary term. The font value is linked to the left_indent, right_indent, pre_skip, post_skip, skip and align attributes of the **:gl** tag (see "Font Linkage" on page 77).

12.3.38 HEADING

Specify information which applies to headings in general.

```
:HEADING
          delim = '.'
          stop_eject = no
          para_indent = no
          threshold = 2
          max_group = 10
```

delim The delim attribute sets the heading number delimiter to a specific character.

stop_eject This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, a heading which would force the beginning of a new page will not cause a page ejection if it immediately follows another heading.

para_indent This attribute accepts the keyword values *yes* and *no*. If the value *no* is specified, the indentation of the first line in a paragraph after a heading is suppressed.

threshold This attribute accepts as a value a non-negative integer number. The specified value indicates the minimum number of text lines which must fit on the page. The heading will be forced to the next page or column if the threshold requirements are not met by the following document element. The threshold attribute of the heading overrides the default threshold specified by the **:widow** tag.

max_group This attribute accepts a positive integer number. If a group of headings are forced to a new page or column because of threshold requirements, the specified value will limit the number of headings forced as a group.

12.3.39 Hn

Define the characteristics of a heading tag, where **n** is between zero and six inclusive.

```
:H0
      group = 0
      indent = '0.5i'
      pre_top_skip = 4
      pre_skip = 0
      post_skip = 4
      spacing = 1
      font = 3
      number_font = 3
      number_form = none
      page_position = left
      number_style = h
      page_eject = yes
      line_break = yes
      display_heading = yes
      number_reset = yes
      case = mixed
      align = 0
```

group The group attribute accepts any non-negative number between 0 and 9. The group value determines which set of headings are processed bye the heading. tags/control words.

indent The indent attribute accepts any valid horizontal space unit. The indent value is added to the offset determined by the page position attribute, giving the starting offset from the left margin for the heading.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the heading. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the heading. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the heading. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the heading if it takes more than one line.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the heading text. The font value is linked to the indent,

pre_top_skip and post_skip attributes (see "Font Linkage" on page 77).

number_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The number font attribute defines the font of the heading number.

number_form This attribute accepts the values *none*, *prop*, and *new*. The specified value determines the format of the heading number. The value *none* indicates that no number is to be output. The value *prop* indicates that the number is composed of the number for the current level prefixed by the number for the previous level and the number delimiter specified in the **heading** layout tag. The value *new* indicates that only the number of the current level is to be output.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the heading between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

number_style This attribute sets the number style of the heading number. (See "Number Style" on page 121).

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

line_break This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the skip value specified by the post_skip attribute

will be issued. If the value *no* is specified, the skip value specified by the post_skip attribute will be ignored. If a paragraph follows the heading, the paragraph text will start on the same line as the heading.

display_heading This attribute accepts the keyword values *yes* and *no*. If the value *no* is specified, the heading line will not be displayed. The heading will still be internally created, and used in the table of contents.

number_reset This attribute accepts the keyword values *yes* and *no*. When a heading is processed, all heading levels after it have their heading numbers reset. When the value 'no' is specified, the number of the next level of heading is not reset.

case This attribute accepts the keyword values *mixed*, *upper* and *lower*. When a heading is processed, the text is converted to upper or lower case when the values UPPER or LOWER are used. The text is left unchanged when the value MIXED is used.

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the heading. After the heading is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the heading.

12.3.40 INDEX

Define the characteristics of the index section.

```
:INDEX
         post_skip = 0
         pre_top_skip = 0
         left_adjust = 0
         right_adjust = 0
         spacing = 1
         columns = 1
         see_string = "See "
         see_also_string = "See also "
         header = no
         index_string = "Index"
         page_eject = yes
         page_reset = no
         font = 1
```

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after

the heading. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page. If the index heading is not displayed (the header attribute has a value of NO), the post-skip value has no effect.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the heading. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the index.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the index.

see_string This attribute accepts a character string. The specified string precedes any *see* text generated in the index.

see_also_string This attribute accepts a character string. The specified string precedes any *see also* text generated in the index.

header The header attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the index heading is generated. If the value *no* is specified, the header text is not generated.

index_string This attribute accepts a character string. If the index header is generated, the specified string is used for the heading text.

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

page_reset This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the page number is reset to one at the beginning of the section.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the header attribute value. The font value is linked to the left_adjust, right_adjust, pre_top_skip and post_skip attributes (see "Font Linkage" on page 77).

12.3.41 IXHEAD

Define the characteristics of the index headings. In most cases, the index heading is the letter which starts the index terms following it.

```
:IXHEAD
          pre_skip = 2
          post_skip = 0
          font = 2
          indent = 0
          frame = box
          header = yes
```

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the index heading. The pre-skip will be merged with the previous

document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the index heading. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the index heading. The font value is linked to the indent, pre_skip and post_skip attributes (see "Font Linkage" on page 77).

indent The indent attribute accepts any valid horizontal space unit. The attribute space value is added to the current left margin before the index heading is generated in the index. The left margin is reset to its previous value after the heading is generated.

frame This attribute accepts the values *rule*, *box*, *none*, and *'character string'*. The specified attribute value determines the type of framing around the index heading. See the discussion of the frame attribute under "FIG" on page 92 for an explanation of the attribute values.

header This attribute accepts the keyword values *yes* and *no*. If 'no' is specified, the index heading is not displayed. The font and frame attributes are ignored, and the pre and post skip values are merged.

12.3.42 IXMAJOR

Define the characteristics of the major index signifiers.

```
:IXMAJOR
          font = 2
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the

highlighting phrase GML tags. The font attribute defines the font of the major indexing signifier.

12.3.43 IXPGNUM

Define the characteristics of the index page numbers.

```
:IXPGNUM
          font = 0
```

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the index page numbers.

12.3.44 In

Define the characteristics of an index entry level, where *n* is 1, 2, or 3. The *string_font* attribute is only valide with index entry levels one and two.

```
:I1
      pre_skip = 1
      post_skip = 1
      skip = 1
      font = 0
      indent = 0
      wrap_indent = '0.4i'
      index_delim = "  "
      string_font = 0
```

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the index entry. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the index entry. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of

an output page, any remaining part of the skip is not carried over to the next output page.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between each entry in an index level.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the index entry. The font value is linked to the wrap_indent, skip, pre_skip and post_skip attributes (see "Font Linkage" on page 77).

indent The indent attribute accepts any valid horizontal space unit. The attribute space value is added to the current left margin before the index entry is produced in the index. After the index entries under the current entry are produced, the left margin is reset to its previous value.

wrap_indent This attribute accepts as a value any valid horizontal space unit. If the list of references for an index entry in the index does not fit on one output line, the specified attribute value indicates the indentation that is to occur on the following output lines.

index_delim This attribute accepts a string value which is placed between the index text and the index page number(s). If the text, page number(s) and delimiter does not fit on one output line, the delimiter text is not used.

string_font This attribute accepts a positive integer number, and is valid with the **:i1** and **:i2** layout tags. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to highlighting levels specified by the highlighting phrase GML tags. The string_font attribute defines the font of the the *see* and *see_also* attribute strings defined by the :INDEX layout tag.

12.3.45 LAYOUT

Start the layout definition.

```
:LAYOUT
```

12.3.46 LETDATE

Define the characteristics of the date entity in the letter tag format.

```
:LETDATE
          date_form = "$ml $dsn, $yl"
          depth = 15
          font = 0
          page_position = right
```

date_form The date_form attribute accepts a character string value which defines the format of the date string. The year, month and day may be specified separately and in any order by special date sequences. These date sequences are started with a dollar($) sign and followed by one to three characters. Text which is not recognized as a date sequence can be entered to tailor the format of the resulting date.

The first character in a date sequence is a *Y* for the year, an *M* for the month, or a *D* for the day. The next character is the *L* or *S* character to specify the long or short form of the date sequence. If neither of these characters are present, the long form is used. When the length specifier is present, the *N* character is used to format the month or the day as a number. If the length specified is not present, the month and day values are created in character form. The year is always formatted as a number.

depth The depth attribute specifies the amount of space to leave before the line of date text. This attribute accepts any valid vertical space unit. This attribute is often used to position beyond pre-printed letter head.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the date text. The font value is linked to the date_form and depth attributes (see "Font Linkage" on page 77).

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the date between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output

next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

12.3.47 LP

Define the characteristics of the list part entity.

```
:LP
      left_indent = 0
      right_indent = 0
      line_indent = 0
      pre_skip = 1
      post_skip = 1
      spacing = 1
```

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the list part.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the list part.

line_indent The line_indent attribute accepts any valid horizontal space unit. This attribute specifies the amount of indentation for the first output line of the list part.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the list part. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the list part. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text

lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the list part.

12.3.48 LQ

Define the characteristics of the long quote entity.

```
:LQ
      left_indent = '0.25i'
      right_indent = '0.25i'
      pre_skip = 1
      post_skip = 1
      spacing = 1
      font = 0
```

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the long quote.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the long quote.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the long quote. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the long quote. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in

the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the long quote.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the quote text. The font value is linked to the left_indent, right_indent, pre_skip and post_skip attributes (see "Font Linkage" on page 77).

12.3.49 NOTE

Define the characteristics of the note entity.

```
:NOTE
        left_indent = 0
        right_indent = 0
        pre_skip = 1
        post_skip = 1
        font = 2
        spacing = 1
        note_string = "NOTE: "
```

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the note.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the note.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the note. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after

the note. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the text specified by the *note_string* attribute. The font value is linked to the left_indent, right_indent, pre_skip and post_skip attributes (see "Font Linkage" on page 77).

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the note.

note_string This attribute accepts a character string. The specified string precedes the text of the note. The length of this string determines indentation of the note text.

12.3.50 OL

Define the characteristics of the ordered list entity.

```
:OL
      level = 1
      left_indent = 0
      right_indent = 0
      pre_skip = 1
      skip = 1
      spacing = 1
      post_skip = 1
      font = 0
      align = '0.4i'
      number_style = hd
      number_font = 0
```

level This attribute accepts a positive integer number. If not specified, a level value of '1'. is assumed. Each list level is separately specified. For example, if two levels of the ordered list are specified, the **:ol** tag will be specified twice in the layout. When some attributes for a new level of a list are not specified, the default

values for those attributes will be the values of the first level. Since list levels may not be skipped, each new level of list must be sequentially defined from the last specified level.

If there is an ordered, simple, and second ordered list nested together in the document, the simple and first ordered list will both be from level one, while the last ordered list will be level two. The appropriate level number is selected based on the nesting level of a particular list type. If a list type is nested beyond the levels specified in the layout, the levels are "cycled". For example, if there are two levels of ordered list specified in the layout, and there are three ordered lists nested, the third level of ordered list will use the attributes of the level one ordered list. A fourth nested list would use the attributes of the level two.

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the ordered list.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the ordered list.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the ordered list. The pre-skip will be merged with the previous document entity’s post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between list items.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the list item.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the ordered list. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the ordered list text. The font value is linked to the left_indent, right_indent, pre_skip, post_skip and skip attributes (see "Font Linkage" on page 77).

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the list item number. After the list item number is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the list item.

number_style This attribute sets the number style of the list item number. (See "Number Style" on page 121).

number_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The number font attribute defines the font of the list item number. The font value is linked to the align attibute (see "Font Linkage" on page 77).

12.3.51 OPEN

Define the characteristics of the open entity in the letter tag format.

```
:OPEN
        pre_top_skip = 2
        font = 0
        delim = ':'
```

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the opening salutation. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the opening salutation. The font value is linked to the pre_top_skip attribute (see "Font Linkage" on page 77).

delim The delim attribute sets the delimiter that is output following the opening salutation to a specific character.

12.3.52 P

Define the characteristics of the paragraph entity.

```
:P
     line_indent = 0
     pre_skip = 1
     post_skip = 0
```

line_indent The line_indent attribute accepts any valid horizontal space unit. This attribute specifies the amount of indentation for the first output line of the paragraph.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the paragraph. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the paragraph. The post-skip will be merged with the next

document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

12.3.53 PAGE

Define the boundaries of the document on the output page.

```
:PAGE
        top_margin = 0
        left_margin = '1i'
        right_margin = '7i'
        depth = '9.66i'
```

top_margin The top margin attribute specifies the amount of space between the top of the page and the start of the output text. This attribute accepts any valid vertical space unit.

left_margin The left margin attribute specifies the amount of space between the left side of the page and the start of the output text. This attribute accepts any valid horizontal space unit.

right_margin The right margin attribute specifies the amount of space between the left side of the page and the right margin of of the output text. This attribute accepts any valid horizontal space unit.

depth The depth attribute specifies the depth of the output page. Output text starts at the top margin and ends at the bottom margin of the page. The bottom margin is the sum of the *top_margin* and *depth* attribute values. This attribute accepts any valid vertical space unit.

12.3.54 PC

Define the characteristics of the paragraph continuation entity.

```
:PC
      line_indent = 0
      pre_skip = 1
      post_skip = 0
```

line_indent The line_indent attribute accepts any valid horizontal space unit. This attribute specifies the amount of indentation for the first output line of the paragraph continuation.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the paragraph continuation. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the paragraph continuation. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

12.3.55 PREFACE

Define the characteristics of the preface section and preface heading.

```
:PREFACE
          post_skip = 1
          pre_top_skip = 1
          font = 1
          spacing = 1
          header = yes
          preface_string = "PREFACE"
          page_eject = yes
          page_reset = no
          columns = 1
```

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the preface heading. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page. If the preface heading is not displayed (the header attribute has a value of NO), the post-skip value has no effect.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before

the preface heading. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the preface heading. The font value is linked to the pre_top_skip and post_skip attributes (see "Font Linkage" on page 77).

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the preface.

header The header attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the preface heading is generated. If the value *no* is specified, the header text is not generated.

preface_string This attribute accepts a character string. If the preface header is generated, the specified string is used for the heading text.

page_eject This attribute accepts the keyword values *yes*, *no*, *odd*, and *even*. If the value *no* is specified, the heading is one column wide and is not forced to a new page. The heading is always placed on a new page when the value *yes* is specified. Values other than *no* cause the heading to be treated as a page wide heading in a multi-column document.

The values *odd* and *even* will place the heading on a new page if the parity (odd or even) of the current page number does not match the specified value. When two headings appear together, the attribute value *stop_eject=yes* of the **:heading** layout tag will normally prevent the the second heading from going to the next page. The *odd* and *even* values act on the heading without regard to the *stop_eject* value.

page_reset This attribute accepts the keyword values *yes* and *no*. If the value *yes* is specified, the page number is reset to one at the beginning of the section.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the preface.

12.3.56 SAVE

Save the current layout into the specified file name. This tag is equivalent to the **:convert** tag.

```
:SAVE file='filename'.
```

12.3.57 SL

Define the characteristics of the simple list entity.

```
:SL
      level = 1
      left_indent = 0
      right_indent = 0
      pre_skip = 1
      skip = 1
      spacing = 1
      post_skip = 1
      font = 0
```

level This attribute accepts a positive integer number. If not specified, a level value of '1'. is assumed. Each list level is separately specified. For example, if two levels of the ordered list are specified, the **:sl** tag will be specified twice in the layout. When some attributes for a new level of a list are not specified, the default values for those attributes will be the values of the first level. Since list levels may not be skipped, each new level of list must be sequentially defined from the last specified level.

If there is an ordered, simple, and second ordered list nested together in the document, the simple and first ordered list will both be from level one, while the last ordered list will be level two. The appropriate level number is selected based on the nesting level of a particular list type. If a list type is nested beyond the levels specified in the layout, the levels are "cycled". For example, if there are two levels of ordered list specified in the layout, and there are three ordered lists nested, the third level of ordered list will use the attributes of the level one ordered list. A fourth nested list would use the attributes of the level two.

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the simple list.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the simple list.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the simple list. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between list items.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the list items.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the simple list. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the simple list text. The font value is linked to the left_indent,

right_indent, pre_skip, post_skip and skip attributes (see "Font Linkage" on page 77).

12.3.58 SUBJECT

Define the characteristics of the subject entity in the letter tag format.

```
:SUBJECT
          left_adjust = 0
          page_position = centre
          pre_top_skip = 2
          font = 1
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the subject line between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the subject line. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the subject line. The font value is linked to the left_adjust and pre_top_skip attributes (see "Font Linkage" on page 77).

12.3.59 TITLE

Define the characteristics of the title line entity.

```
:TITLE
         left_adjust = 0
         right_adjust = '1i'
         page_position = right
         font = 2
         pre_top_skip = 15
         skip = 2
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the title line between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the title line. The font value is linked to the left_adjust, right_adjust, pre_top_skip and skip attributes (see "Font Linkage" on page 77).

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the title lines. The pre-top-skip will be merged with the previous document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between title lines.

12.3.60 TITLEP

Define the characteristics of the title part section.

```
:TITLEP
          spacing = 1
          columns = 1
```

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the title part.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the title part section.

12.3.61 TO

Define the characteristics of the TO entity in the letter tag format.

```
:TO
      left_adjust = 0
      page_position = left
      pre_top_skip = 1
      font = 0
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

page_position This attribute accepts the values *left*, *right*, *center*, and *centre*. The position of the TO text between the left and right margins is determined by the value selected. If *left* is the attribute value, the text is output at the left margin. If *right* is the attribute value, the text is output next to the right margin. When *center* or *centre* is specified, the text is centered between the left and right margins.

pre_top_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the TO text. The pre-top-skip will be merged with the previous

document entity's post-skip value. The specified space is still skipped at the beginning of a new page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the TO text. The font value is linked to the left_adjust and pre_top_skip attributes (see "Font Linkage" on page 77).

12.3.62 TOC

Define the characteristics of the table of contents.

```
:TOC
       left_adjust = 0
       right_adjust = 0
       spacing = 1
       columns = 1
       toc_levels = 4
       fill_string = "."
```

left_adjust The left_adjust attribute accepts any valid horizontal space unit. The left margin is set to the page left margin plus the specified left adjustment.

right_adjust The right_adjust attribute accepts any valid horizontal space unit. The right margin is set to the page right margin minus the specified right adjustment.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the table of contents.

columns The columns attribute accepts a positive integer number. The columns value determines how many columns are created for the table of contents.

toc_levels This attribute accepts as its value a positive integer value. The attribute value specifies the maximum level of the entries that appear in the table of contents. For example, if the attribute value is

four, heading levels zero through three will appear in the table of contents.

fill_string This attribute accepts a string value which is used to 'fill' the line between the text and the page number.

12.3.63 TOCHn

Define the characteristics of a table of contents heading, where *n* is between zero and six inclusive.

```
:TOCH0
        group = 0
        indent = 0
        skip = 1
        pre_skip = 1
        post_skip = 1
        font = 0
        align = 0
        display_in_toc = yes
```

group The group attribute accepts any non-negative number between 0 and 9. The group value determines which set of table of contents are processed bye the group of level **n** table of contents heading entries. tags/control words.

indent The indent attribute accepts any valid horizontal space unit. The attribute space value is added to the current left margin before the table of contents entry is produced. After all of the subentries under the current entry are produced, the left margin is reset to its previous value.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between the individual entries within the group of level **n** table of contents heading entries.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the group of level **n** table of contents heading entries. The pre-skip will be merged with the previous document entity's post-skip value.

If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the group of level **n** table of contents heading entries. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the table of contents heading. After the table of contents heading is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the heading entry.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the heading produced in the table of contents entry.

display_in_toc This attribute accepts the keyword values *yes* and *no*. The heading for the table of contents entry is not produced when the value *no* is specified. The entries pre and post skips are still generated.

12.3.64 TOCPGNUM

Define the characteristics of the table of contents page numbers.

```
:TOCPGNUM
            size = '0.4i'
            font = 0
```

size This attribute accepts any valid horizontal space unit. The specified value is the minimum amount of space that will be reserved on the output line for the page number of a table of contents entry.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the

highlighting phrase GML tags. The font attribute defines the font of the page number. The font value is linked to the size attribute (see "Font Linkage" on page 77).

12.3.65 UL

Define the characteristics of the unordered list entity.

```
:UL
      level = 1
      left_indent = 0
      right_indent = 0
      pre_skip = 1
      skip = 1
      spacing = 1
      post_skip = 1
      font = 0
      align = '0.4i'
      bullet = '*'
      bullet_translate = yes
      bullet_font = 0
```

level This attribute accepts a positive integer number. If not specified, a level value of '1'. is assumed. Each list level is separately specified. For example, if two levels of the ordered list are specified, the **:ul** tag will be specified twice in the layout. When some attributes for a new level of a list are not specified, the default values for those attributes will be the values of the first level. Since list levels may not be skipped, each new level of list must be sequentially defined from the last specified level.

If there is an ordered, simple, and second ordered list nested together in the document, the simple and first ordered list will both be from level one, while the last ordered list will be level two. The appropriate level number is selected based on the nesting level of a particular list type. If a list type is nested beyond the levels specified in the layout, the levels are "cycled". For example, if there are two levels of ordered list specified in the layout, and there are three ordered lists nested, the third level of ordered list will use the attributes of the level one ordered list. A fourth nested list would use the attributes of the level two.

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the unordered list.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the unordered list.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the unordered list. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped between list items.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the list items.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the unordered list. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the unordered list text. The font value is linked to the left_indent, right_indent, pre_skip, post_skip and skip attributes (see "Font Linkage" on page 77).

align This attribute accepts any valid horizontal space unit. The align value specifies the amount of space reserved for the list item bullet. After the list item bullet is produced, the align value is added to the current left margin. The left margin will be reset to its previous value after the list item.

bullet This attribute specifies the single character value which annotates an unordered list item.

bullet_translate This attribute accepts the keyword values *yes* and *no*. If 'yes' is specified, input translation is performed on the annotation character.

bullet_font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The bullet_font attribute defines the font of the annotation character. The font value is linked to the align attibute (see "Font Linkage" on page 77).

12.3.66 WIDOW

Define the widowing control of document elements.

```
:WIDOW
         threshold = 2
```

threshold This attribute accepts as a value a non-negative integer number. The specified value indicates the minimum number of text lines which must fit on the page. A document element will be forced to the next page or column if the threshold requirement is not met.

12.3.67 XMP

Define the characteristics of the example entity.

```
:XMP
       left_indent = '0.25i'
       right_indent = 0
       pre_skip = 2
       post_skip = 0
       spacing = 1
       font = 0
```

left_indent This attribute accepts any valid horizontal space unit. The left indent value is added to the current left margin. The left margin will be reset to its previous value at the end of the example.

right_indent This attribute accepts any valid horizontal space unit. The right indent value is subtracted from the current right margin. The right margin will be reset to its previous value at the end of the example.

pre_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped before the example. The pre-skip will be merged with the previous document entity's post-skip value. If a pre-skip occurs at the beginning of an output page, the pre-skip value has no effect.

post_skip This attribute accepts vertical space units. A zero value means that no lines are skipped. If the skip value is a line unit, it is multiplied by the current line spacing (see "Vertical Space Unit" on page 77 for more information). The resulting amount of space is skipped after the example. The post-skip will be merged with the next document entity's pre-skip value. If a post-skip occurs at the end of an output page, any remaining part of the skip is not carried over to the next output page.

spacing This attribute accepts a positive integer number. The spacing determines the number of blank lines that are output between text lines. If the line spacing is two, each text line will take two lines in the output. The number of blank lines between text lines will therefore be the spacing value minus one. The spacing attribute defines the line spacing within the example.

font This attribute accepts a non-negative integer number. If a font number is used for which no font has been defined, WATCOM Script/GML will use font zero. The font numbers from zero to three correspond directly to the highlighting levels specified by the highlighting phrase GML tags. The font attribute defines the font of the example. The font value is linked to the left_indent, right_indent, pre_skip and post_skip attributes (see "Font Linkage" on page 77).

13 GML Summary

```
general elements
pre GDOC elements
:GDOC sec='classification'
      (sec is optional)
general elements
  :FRONTM.
    :TITLEP.
      :TITLE.title
      :DOCNUM.document number
      :DATE.date
      :AUTHOR.author's name
      :ADDRESS.
        :ALINE.address line
      :eADDRESS.
    :eTITLEP.
    :ABSTRACT.
      basic document elements
      (headings H2-H6 allowed)
    :PREFACE.
      basic document elements
      (headings H2-H6 allowed)
    :TOC.
    :FIGLIST.
  :BODY.
    basic document elements
    (headings H0-H6 allowed)
  :APPENDIX.
    basic document elements
    (headings H1-H6 allowed)
  :BACKM.
    basic document elements
    (headings H1-H6 allowed)
    :INDEX.
:eGDOC.
```

Figure 76. *Overall Document Structure*

13.1 Front Material

```
:FRONTM.
  title page
  abstract
  preface
  table of contents
  list of figures
```

13.1.1 Title Page

```
:TITLEP.
  :TITLE stitle = short title.title text
        (stitle is optional)
  :DOCNUM.document number
  :DATE.date text
       (date text is optional)
  :AUTHOR.author's name
  :ADDRESS.
    :ALINE.address line
      (may occur several times)
  :eADDRESS.
:eTITLEP.
```

13.1.2 Abstract

```
:ABSTRACT.
  basic document elements
  (headings H2-H6 allowed)
```

13.1.3 Preface

```
:PREFACE.
  basic document elements
  (headings H2-H6 allowed)
```

13.1.4 Table of Contents

```
:TOC.
```

13.1.5 List of Figures

```
:FIGLIST.
```

13.2 Body

```
:BODY.
  basic document elements
  (headings H0-H6 allowed)
```

13.3 Appendix

```
:APPENDIX.
  basic document elements
  (headings H1-H6 allowed)
```

13.4 Back Material

```
:BACKM.
  basic document elements
  (headings H1-H6 allowed)
  index
```

13.4.1 Index

```
:INDEX.
```

13.5 Basic Document Elements

```
Address              Long Quotation
Definition List      Note
Example              Ordered List
Figure               Paragraph
Footnote             Paragraph Continuation
Glossary List        Simple List
Graphic              Unordered List
```

13.6 Paragraph Elements

May contain one or more lines of text and/or any of the following:

Citation
Figure Reference
Footnote Reference
Heading Reference
List Item Reference
Highlighted Phrase
Quote
Set Font

13.7 Definitions

Character String (char-string) A character string to be associated with an attribute.

Choices(0|1|2) The bar (|) separates the various choices that may be associated with the attribute.

Identifier Name (id-name) Seven character name consisting of letters and numbers.

Symbol Name Maximum of ten characters consisting of letters, numbers, and the characters @, #, $ and underscore(_) character.

Text Line One line of text to be processed as defined by the tag.

13.8 Examples and Figures

13.8.1 Example

```
:XMP depth = 'vert-space-unit'.
     (depth is optional)
   paragraph elements
   basic document elements
:eXMP.
```

13.8.2 Figure

```
:FIG attribute.
   paragraph elements
   basic document elements
:FIGCAP.caption text
   (figcap is optional)
:FIGDESC.
   (figdesc is optional)
     paragraph elements
     basic document elements
:eFIG.
```

The attribute, if specified, is one or more of

depth = 'vert-space-unit'
frame = box | rule | none | 'char-string'
id = 'id-name'
place = top | bottom | inline
width = page | column | 'hor-space-unit'

13.8.3 Figure Reference

```
:FIGREF refid = 'id-name' page = yes | no .
                          (page is optional)
```

13.9 Headings

13.9.1 Heading

```
:Hn attribute.text line
```

The value of *n* must be one of 0, 1, 2, 3, 4, 5, or 6. The attribute, if specified, is one or more of

id = 'id-name'
stitle = 'char-string'

The *stitle* attribute is only permitted if *n* is 0 or 1.

13.9.2 Heading Reference

```
:HDREF refid = 'id-name' page = yes | no .
                          (page is optional)
```

13.10 Lists

13.10.1 Address

```
:ADDRESS.
:ALINE.address line
   (may occur several times)
:eADDRESS.
```

13.10.2 Definition List

```
:DL attribute.
   :DTHD.text line
      (may occur several times)
   :DDHD.text line
      (for every DTHD there must be a DDHD)
   :DT.text line
      (may occur several times)
   :DD.
       paragraph elements
       basic document elements
      (for every DT there must be a DD)
   :LP.
       paragraph elements
      (LP is optional)
:eDL.
```

The attribute if specified, may be one or more of

break
compact
headhi = 'integer number'
termhi = 'integer number'
tsize = 'hor-space-unit'

13.10.3 Glossary List

```
:GL attribute.
   :GT.text line
      (may occur several times)
   :GD.
       paragraph elements
       basic document elements
      (for every GT there must be a GD)
   :LP.
       paragraph elements
      (LP is optional)
:eGL.
```

The attribute if specified, may be one or more of

compact
termhi = 0 | 1 | 2 | 3

13.10.4 Ordered List

```
:OL compact.
    (compact is optional)
   :LI id = 'id-name'.
      (id is optional)
      (LI may occur several times)
       paragraph elements
       basic document elements
   :LP.
       paragraph elements
      (LP is optional)
:eOL.
```

13.10.5 Simple List

```
:SL compact.
    (compact is optional)
   :LI id = 'id-name'.
      (id is optional)
      (LI may occur several times)
       paragraph elements
       basic document elements
   :LP.
       paragraph elements
      (LP is optional)
:eSL.
```

13.10.6 Unordered List

```
:UL compact.
    (compact is optional)
   :LI id = 'id-name'.
      (id is optional)
      (LI may occur several times)
       paragraph elements
       basic document elements
   :LP.
       paragraph elements
      (LP is optional)
:eUL.
```

13.10.7 List Reference

```
:LIREF refid = 'id-name' page = yes | no .
                          (page is optional)
```

13.11 Notes

13.11.1 Footnote

```
:FN id = 'id-name'.
   (id is optional)
   paragraph elements
   basic document elements
:eFN.
```

13.11.2 Footnote Reference

```
:FNREF refid = 'id-name'.
```

13.11.3 Note

```
:NOTE.
   paragraph elements
```

13.12 Paragraphs

13.12.1 Paragraph

```
:P.
   paragraph elements
```

13.12.2 Paragraph Continuation

```
:PC.
   paragraph elements
```

13.13 Quotes and Highlighted Phrases

13.13.1 Citation

```
:CIT.
    paragraph elements
:eCIT.
```

13.13.2 Highlighted Phrase

```
:HPn.
   paragraph elements
:eHPn.
```

The value of *n* must be one of 0, 1, 2, or 3

13.13.3 Long Quotation

```
:LQ.
   basic document elements
:eLQ.
```

13.13.4 Quote

```
:Q.
   paragraph elements
:eQ.
```

13.13.5 Set Font

```
:SF font=n.
   paragraph elements
:eSF.
```

13.14 Graphics

```
:GRAPHIC file = 'char-string'
         depth = 'vert-space-unit'
         width = 'hor-space-unit'
         scale = integer number
         xoff = 'hor-space-unit'
         yoff = 'vert-space-unit'.
```

The *file* attribute must always be specified. The *depth* attribute is required if the graphic file is in the PostScript format.

13.15 General Elements

General Elements may appear any place in the document source.

13.15.1 Comment

```
:CMT.text line
```

13.15.2 Include

```
:INCLUDE file = 'char-string'.
```

13.15.3 Set

```
:SET symbol = 'symbol-name'
     value = 'char-string' | delete .
```

13.16 Pre GDOC Elements

Pre GDOC Elements may appear any place in the document source before the **:GDOC** tag.

13.16.1 Imbedding Layouts

This tag is equivalent to the **:include** tag.

```
:IMBED file = 'char-string'.
```

13.16.2 Defining Layouts

```
:LAYOUT.
   layout tags
   :SAVE file='file-name'.
   :CONVERT file='file-name'.
:eLAYOUT.
```

13.17 Post GDOC Elements

Post GDOC Elements may appear any place in the document source after the **:GDOC** tag.

13.17.1 Binary Include

```
:BINCLUDE file = 'char-string'
          reposition = start|end
          depth = 'vert-space-unit'.
```

13.17.2 Index Entries

```
:In attribute.text line
```

The value of *n* must be 1, 2 or 3. The attribute, if specified, may be one or more of

id = 'id-name'
pg = start | end | major | 'char-string'
refid = 'id-name' (only with I2 or I3)

13.17.3 Index Header

```
:IHn attribute.text line
```

The value of *n* must be 1, 2 or 3. The attribute, if specified, may be one or more of

id = 'id-name'
print = 'char-string'
see = 'char-string' (only with IH1 or IH2)
seeid = 'id-name' (only with IH1 or IH2)

13.17.4 Index Reference

```
:IREF refid = 'id-name' attribute.
```

The attribute, if specified, may be one or more of

pg = start | end | major | 'char-string'
see = 'char-string' (only when referencing IH1 or IH2)
seeid = 'id-name' (only when referencing IH1 or IH2)

13.17.5 Process Specific Control

```
:PSC proc = 'char-string'.
     (proc is optional)
:ePSC.
```

13.18 WATCOM Letter Format

```
:GDOC sec = 'classification'.
      (sec is optional)
   :FROM.
      address line
        (may occur several times)
   :DATE align = 'value'
         depth = 'vert-space-unit'.date-text
        (align and depth are optional)
        (date-text is optional)
   :DOCNUM.document number
   :TO compact.
        (compact is optional)
      Recipient-line
        (may occur several times)
   :ATTN.attention name
   :SUBJECT.subject text
   :OPEN.opening salutation
      basic document elements
   :CLOSE depth = 'vert-space-unit'.closing salutation
          (depth is optional)
      author line
        (may occur several times)
   :eCLOSE.typist mark
          (typist mark is optional)
   :DISTRIB.
   :DIST.label
      names
        (one per line)
   :eDISTRIB.
:eGDOC.
```

14 Running WATCOM Script/GML

This chapter describes how you invoke WATCOM Script/GML and the options that may be specified. The subsections provide information which is specific to each of the systems supported by WATCOM Script/GML, followed by a description of the available options.

WATCOM Script/GML is invoked by entering:

```
WGML file-name options
```

The "file-name" specifies the file containing the source text and GML tags for the document. If the file type part of the file name (see "Files" on page 281) is not specified, WATCOM Script/GML searches for source files with the alternate file extension followed by the file type of *GML*. When a file type is specified, WATCOM Script/GML searches for source files with that file type.

It is possible that many command line options will be necessary to process a document. The *command file* support provided by WATCOM Script/GML allows you to place these options in a file. The options in the command file are processed by specifying the *file* option on the WATCOM Script/GML command line.

The option file "default" is located and loaded before other options are processed. The search path for the default option file is the current disk location, the device library path, followed by the document include path.

NOTE: The *device* option must always be specified.

14.1 Command Lines with IBM VM/CMS and IBM PC/DOS

The options are separated from the file name by a left parenthesis. Options and their values are separated from each other by a space character. For example,

```
WGML book ( DEVICE qume
```

If an option value contains a space, it should be enclosed in double(") or single(') quotes. For example,

```
WGML book ( DEVICE "qume device" DELIM !
```

If the options will not fit on one line, they may be continued to a new input line by omitting the source file name. WATCOM Script/GML will request command line input until a file name is specified. The left parenthesis must precede the options on each of the entered command lines to differentiate them from a file name.

```
WGML ( DEVICE qume
book ( DELIM !
```

Options such as the font option require more than one value for each font being specified. These options accept a list of values. Each value in the list is separated by a space. An option value list may be continued to a new input line.

```
WGML ( DEVICE qume DELIM ! FONT 0 mono10
book ( bold FONT 1 mono12 bold
```

The above example overrides two of the default fonts.

14.1.1 Command Files

If a number of options must be used to process a GML document, they can be placed in a command file to reduce the amount of typing each time you process the document. Each line in the command file is entered as if specified at the terminal. The default file type for a WATCOM Script/GML command file is **gmlcmd** with IBM VM/CMS, and **opt** with IBM PC/DOS. (For more information, see "FILE" on page 213).

```
( FONT 0 mono10 plain
( FONT 1 mono12 bold
```

If the lines in the above example are in a file with the name "setfont" (with a file type of gmlcmd or opt), the following example shows how the command file is referenced.

```
WGML ( DEVICE qume FILE setfont
book ( DELIM !
```

The options in the file "setfont" are processed with those options specified on the command line. Note that each line in the command file begins with a left parenthesis. This allows the specification of a file name on the WATCOM Script/GML command line.

14.1.2 IBM VM/CMS Specifics

The device library must be specified as a global maclib (see "Libraries with IBM VM/CMS" on page 294). The following default file types are use by WATCOM Script/GML:

File Type	*Usage*
GML	document source files
LAYOUT	layout files created with the **:save** tag
GMLCMD	command files
VALUES	value files specified by the VALUESET command line option

14.1.3 IBM PC/DOS Specifics

The DOS environment symbol **GMLLIB** must be defined for locating device information used by the WGML and WGMLUI programs. (see "Libraries with IBM PC/DOS" on page 297).

When WATCOM Script/GML is processing a document, some internal data may be store on disk. Setting the DOS symbol **GMLPAG** with a path list directs the disk locations used to store this information.

```
SET GMLPAG=E:;C:\
```

The set in the previous example directs WATCOM Script/GML to use the `E:\` disk and root directory. When the disk is full, it will start using the `C:\` disk location. If the `E:` disk drive is a RAM disk, WATCOM Script/GML will process the document in less time.

The WGMLUI program will accept two parameters. Both parameters may be quoted and empty. The first parameter is the name of the document file. The second parameter is the name of the an options file.

The WGMLUI program will locate and invoke the `wedit.exe` program when an edit request is made. If the DOS symbol **EDITNAME** is defined, WGMLUI will use the symbol value as the program name to invoke.

The following default file types are used by WATCOM Script/GML:

File Type	*Usage*
GML	document source files
LAY	layout files created with the **:save** tag

OPT	command files
VAL	value files specified by the VALUESET command line option

14.2 Command Lines with DEC VAX/VMS

The format of the WATCOM Script/GML command line conforms to the standard DEC VAX/VMS format. Documentation available with the system gives a more detailed explanation of this format.

Each option on the WGML command line starts with a slash(/) character. If the option has an associated value, the option and its value are separated by an equal sign. For example,

```
WGML book/DEVICE=qume
```

Some characters, such as slashes and single quotes('), have special meanings for the Digital Command Language(DCL) processor. If the option value contains a special character, it should be enclosed in double quotes("). For example,

```
WGML book/DEVICE=qume/DELIM="!"
```

If the options will not fit on one line, they may be continued to a new input line by entering a hyphen(-) at the end of the line. More options may then be entered on a new line. For example,

```
WGML book/DEVICE=qume -
/DELIM="!"
```

Options such as the font option require more than one value for each font being specified. These options accept a list of values. The value list starts with a left parenthesis and ends with a right parenthesis. Each value in the list is separated by a comma. An option value list may be continued to a new input line, as in the following example:

```
WGML book/DEVICE=qume/DELIM="!"/FONT=(0,mono10,-
plain,"","",1,mono12,bold,"","")
```

There are five values for each font specified. The example above overrides two of the default fonts. Both of the fonts are specified in the same list.

14.2.1 Command Files

If a number of options must be used to process a GML document, they can be placed in a command file to reduce the amount of typing each time you process the document.

Each line in the command file is entered as if specified at the terminal. The default file type for a WATCOM Script/GML command file is **opt**. (For more information, see "FILE" on page 213).

```
/FONT=(0,mono10,plain,-
"","",1,mono12,bold,"","")
```

If the lines in the above example are in the file "setfont.opt", the following example shows how the command file is referenced.

```
WGML book/DELIM="!"/FILE=setfont -
/DEVICE=qume
```

The options in the file "setfont.opt" are processed with those options specified on the command line. Note that with the exception of the last line, all lines in the command file must be continued by ending them with a hyphen.

14.2.2 DEC VAX/VMS Specifics

File Type	*Usage*
GML	document source files
LAY	layout files created with the **:save** tag
OPT	command files
VAL	value files specified by the VALUESET command line option

14.3 Options

The following options may be specified on the WATCOM Script/GML command line. The options are illustrated with an example showing the format for each of the computer systems supported by WATCOM Script/GML. With each option, upper case letters are used to indicate the minimum number of characters that must be specified.

14.3.1 ALTEXTension

```
IBM VM/CMS and IBM PC/DOS
    ALTEXT zgm
DEC VAX/VMS
    /ALTEXT=zgm
```

When a GML source file is specified on the WGML command line, or as an include file, the file type can be omitted. If a source file with the default file type cannot be found, WATCOM Script/GML will search for a file with the file type supplied by the alternate extension option.

14.3.2 Bind

```
IBM VM/CMS and IBM PC/DOS
    BIND odd-margin even_margin
DEC VAX/VMS
    /BIND=(odd-margin,even-margin)
```

The two option values specify the default page margin values for the odd and even pages. If the value for even margin is not specified, the first value applies to both odd and even pages. The initial default value is zero.

14.3.3 CPInch

```
IBM VM/CMS and IBM PC/DOS
    CPI cpi-number
DEC VAX/VMS
    /CPI=cpi-number
```

The characters per inch option specifies the base for determining how much space in the output an integer value represents when used as a horiztontal space unit. The initial value is ’10’.

14.3.4 DELim

```
IBM VM/CMS and IBM PC/DOS
    DEL #
DEC VAX/VMS
    /DEL=#
```

The value of the delimiter option is a single character. The delimiter value is used in the document as the GML tag delimiter in place of the colon character.

14.3.5 DESCription

```
IBM VM/CMS and IBM PC/DOS
    DESC "Layout for Producing a Manual"
DEC VAX/VMS
    /DESC="Layout for Producing a Manual"
```

The DESCRIPTION option specifies a comment in the option file.

14.3.6 DEVice

```
IBM VM/CMS and IBM PC/DOS
    DEV qume
DEC VAX/VMS
    /DEV=qume
```

The DEVICE option must be specified. It determines how the source document is processed to create a formatted document appropriate for the output device being used. Any font definitions previously defined will be deleted.

When working on a PC/DOS system, the DOS environment symbol **GMLLIB** is used to locate the device information (see "Libraries with IBM PC/DOS" on page 297). If the device information is not found, the document include path is searched (see "INCLUDE" on page 102).

14.3.7 DUPlex/NODUPlex

```
IBM VM/CMS and IBM PC/DOS
    DUP
    NODUP
DEC VAX/VMS
    /DUP
    /NODUP
```

The duplex option sets the variable &SYSDUPLEX. to the value "ON". Noduplex will set the value to "OFF".

14.3.8 FILE

```
IBM VM/CMS and IBM PC/DOS
    FILE doqume
DEC VAX/VMS
    /FILE=doqume
```

The specified file is processed as a WATCOM Script/GML command file. This file is composed of options normally specified on the command line. Command files are most useful when many options must be specified, since they provide a way to specify these options without entering them individually each time WATCOM Script/GML is run. Command line records in a command file may include other command file invocations. The default file type for a command file depends on the specific computer system on which WATCOM Script/GML is being run.

When working on a PC/DOS system, the DOS environment symbol **GMLLIB** is used to locate the command file if it is not in the current directory (see "Libraries with IBM PC/DOS" on page 297). If it is still not found, the document include path is searched (see "INCLUDE" on page 102).

14.3.9 FONT

```
IBM VM/CMS and IBM PC/DOS
    FONT font-number font-name font-attribute font-space font-height
DEC VAX/VMS
    /FONT=(font-number,font-name,font-attribute,font-space,font-heig-
ht)
```

The specified font-number is assigned a particular font. The font numbers zero through three correspond to the highlight-phrase tags **:hp0** through **:hp3**. Font numbers greater than three (up to a maximum of 255) may be used in the layout section or with the **:sf** tag.

Each device has a list of available fonts defined with it. The font-name value is selected from these defined fonts, and must be specified.

The font-attribute value specifies an attribute for the defined font. If the font attribute is not specified, the attribute PLAIN is set. The possible values for the font attribute are:

BOLD	The defined font is bolded.
PLAIN	The font is used as defined. This attribute is the default.
ULBOLD	The defined font is bolded and underlined. Spaces are also underlined.
ULINE	The defined font is underlined. Spaces are also underlined.
USBOLD	The defined font is bolded and underscored. Spaces are not underscored.
USCORE	The defined font is underscored. Spaces are not underscored.

When a font is selected for output, the total line height for the font has two components. The first component is the height of the characters in the font, and is fixed by the design of the character set. The second component is a value to create space between lines of a font. Although the line spacing has an optimal value for each font height, it can be modified to adjust the overall look of the document. Both of the line height components are specified as point values (there are 72 points in one inch), with a decimal portion in hundredths of a point (ie. 10.25).

The font-space attribute is optional, and overrides the default space value defined for the font. This attribute may be specified for any type of character font.

The font-height attribute is specified with scaled fonts. Scaled fonts have no predetermined character heights, and must be defined with a height before they can be used. The font-space attribute must also be specified, but can be a null(’’) value to set the default font spacing.

14.3.10 FORMat

```
IBM VM/CMS and IBM PC/DOS
    FORM format-type
DEC VAX/VMS
    /FORM=format-type
```

The FORMAT option specifies the type of GML document to be processed. The format types have different sets of GML tags and, to some extent, a different syntax. The format types available are:

STANDARD	This format provides the standard set of GML tags and is the default.
LETTER	The letter format is designed for processing letters. Refer to "GML Letter Tags" on page 111 for more information.

14.3.11 FROM

```
IBM VM/CMS and IBM PC/DOS
    FROM page-number
DEC VAX/VMS
    /FROM=page-number
```

The FROM option will cause WATCOM Script/GML to print the document starting at the specified page number within the body of the document. The number is specified as an integer, and does not depend on the format of the numbers on the output pages (or even printed on the page). For example, if the page numbers are formatted as roman numerals, the page number you specify would be "4", not "iv". See "TO" on page 221 for more information.

14.3.12 INCList/NOINCList

```
IBM VM/CMS and IBM PC/DOS
    INCL
    NOINCL
DEC VAX/VMS
    /INCL
    /NOINCL
```

The INCLIST option causes WATCOM Script/GML to display on the terminal the name of each source file as it is included in the document. When the NOINCLIST option is specified, the include file names are not displayed on the terminal as they are included. NOINCLIST is the default when in line mode (see "LINEmode" on page 216).

14.3.13 INDex/NOINDex

```
IBM VM/CMS and IBM PC/DOS
    IND
    NOIND
DEC VAX/VMS
    /IND
    /NOIND
```

When the INDEX option is specified, the indexing tags in the document are processed. The **:index** tag must be specified in the back material of the document to produce the index in the output document. The NOINDEX option (the default) will cause the indexing tags in the document to be ignored, creating an empty index.

14.3.14 LAYout

```
IBM VM/CMS and IBM PC/DOS
    LAY file-name
DEC VAX/VMS
    /LAY=file-name
```

A layout file is included at the beginning of the document. This option has the same effect as having an **:include** tag as the first GML source line.

14.3.15 LINEmode

```
IBM VM/CMS and IBM PC/DOS
    LINE
DEC VAX/VMS
    /LINE
```

The presentation of information about the current status of WATCOM Script/GML is displayed on the terminal as separate output lines during the processing of a document. With the IBM PC/DOS system, the full area of the terminal screen is used to display the information. The LINEMODE option forces the display of information from full screen mode to line mode. The options INCLIST, VERBOSE, and STATISTICS are defaulted when in full screen mode. The option WARNING is defaulted in both modes.

14.3.16 LLength

```
IBM VM/CMS and IBM PC/DOS
    LL ll-number
DEC VAX/VMS
    /LL=ll-number
```

The line length option specifies the initial value for the line length used in the document.

14.3.17 LPInch

```
IBM VM/CMS and IBM PC/DOS
    LPI lpi-number
DEC VAX/VMS
    /LPI=lpi-number
```

The lines per inch option specifies the base for determining how much space in the output an integer value represents when used as a vertical space unit. The initial value is '6'.

14.3.18 MAILmerge

```
IBM VM/CMS and IBM PC/DOS
    MAIL file-name
DEC VAX/VMS
    /MAIL=file-name
```

The MAILMERGE option specifies a file containing symbol substitution values or the name of a WATFILE database. This option may be used to create a number of similar documents, such as a form letter mailing. The WATCOM Script/GML processor will inspect the file to determine the file type (WATFILE or symbol values). The following two subsections describe the processing of these files.

14.3.18.1 WATFILE Database File

Each field value in a WATFILE input record is assigned to a symbol name created from the WATFILE field name. If the field name contains characters which are invalid in GML symbol names, the characters up to the invalid character are used.

If the WATFILE file contains the following data:

```
define name                 = 10   L
define addr1                =  6   L
define addr2                =  5   L
bye
John Doe  StreetCity
```

then the three values could be used in the following way:

```
:ADDRESS.
:ALINE.&name.
:ALINE.&addr1.
:ALINE.&addr2.
:eADDRESS.
```

14.3.18.2 Values File

Each record in the values file must contain the same number of symbol values. The document will be produced once for each record in the file. If a layout is specified, it is only processed the first time the document is processed.

Symbol values are separated by commas. Each value may be enclosed in single (') or double (") quotes. The quotation marks surrounding the text are not part of the symbol value. If a quotation mark of the same type used to delimit the symbol value is to be part of the symbol text, it can be entered by specifying the quote character twice. Only one quote character will appear in the resulting symbol value. A symbol value must be quoted if it contains a comma.

The blanks outside quotations, or if the value is not quoted, the blanks before the first and after the last nonblank character, are not considered part of the symbol value. For example, <,,>, <, ,>, <,end_of_record>, or empty records all specify empty symbol values.

Each symbol value in an input record is assigned to a special symbol name. The values are assigned to VALUE1, VALUE2, etc. For example, if the values file contains the following,

```
"John Doe","13 Country Lane","Canada"
```

then the three values could be used in the following way:

```
:ADDRESS.
:ALINE.&value1.
:ALINE.&value2.
:ALINE.&value3.
:eADDRESS.
```

14.3.19 OUTput

```
IBM VM/CMS and IBM PC/DOS
    OUT temp1
DEC VAX/VMS
    /OUT=temp1
```

The output will go to the specified file name instead of the default output file. The default output file name is determined by the device selected on the command line. In some cases the output from WATCOM Script/GML is sent directly to the device, while in other cases the output is sent to a disk file.

If the file name component of the output name is an asterisk, the name of the document is used. For example, an output file specification of `*.ps` when the document name is

`manual.gml` will produce `manual.ps` as the output file. Refer to "OUTPUT_NAME Attribute" on page 265 for more information.

14.3.20 PASSes

```
IBM VM/CMS and IBM PC/DOS
    PASS 2
DEC VAX/VMS
    /PASS=2
```

In some cases WATCOM Script/GML must process a document more than once to properly produce the output document. The value of the passes option is the number of times WATCOM Script/GML must process the document. WATCOM Script/GML will issue a warning message if more passes are necessary. The default passes value is one.

14.3.21 PAUSE/NOPause

```
IBM VM/CMS and IBM PC/DOS
    PAUSE
    NOP
DEC VAX/VMS
    /PAUSE
    /NOP
```

When some of the output devices are selected, information messages are displayed and a response from the keyboard is requested. An example of this is the terminal device which pauses at the bottom of the screen to prevent the output from being scrolled off the screen. The NOPAUSE option suppresses the display of information and requests for keyboard input. PAUSE is the default option.

14.3.22 PROCess

```
IBM VM/CMS and IBM PC/DOS
    PROC x2700
DEC VAX/VMS
    /PROC=x2700
```

The specified name is an alternate condition for the **:psc** tag.

14.3.23 QUIET/NOQuiet

```
IBM VM/CMS and IBM PC/DOS
    QUIET
    NOQ
DEC VAX/VMS
    /QUIET
    /NOQ
```

The quiet option sets the variable &SYSQUIET. to the value "ON". Noquiet will set the value to "OFF".

14.3.24 RESETscreen

```
IBM VM/CMS and IBM PC/DOS
    RESET
DEC VAX/VMS
    /RESET
```

The RESETSCREEN option clears the screen before document processing begins and queries the user when the formatting is complete. Active only when the screen is used in line mode, this option is intended for use when WATCOM Script/GML is invoked from an application program.

14.3.25 SCRipt/NOSCRipt

```
IBM VM/CMS and IBM PC/DOS
    SCR
    NOSCR
DEC VAX/VMS
    /SCR
    /NOSCR
```

The SCRIPT option enables recognition of Script control words and the line separator character. The default option NOSCRIPT will cause these values to be treated as text.

14.3.26 SETsymbol

```
IBM VM/CMS and IBM PC/DOS
    SET processor WGML
DEC VAX/VMS
    /SET=(processor,WGML)
```

The SETSYMBOL option requires two values. The first value is the name of the symbol to be set. The second value is the character string that is to be assigned to the specified symbol name. This option is equivalent to using the **:SET** tag at the beginning of the source document. Refer to "Symbolic Substitution" on page 78 and "SET" on page 107.

14.3.27 STATistics/NOSTATistics

```
IBM VM/CMS and IBM PC/DOS
    STAT
    NOSTAT
DEC VAX/VMS
    /STAT
    /NOSTAT
```

Statistics about the document are displayed after document processing is completed when in line mode, and during the document processing when in full screen mode. Examples of the type of information displayed are the number of input lines processed, the number of include files, and the number of pages produced. NOSTATISTICS is the default when in line mode (see "LINEmode" on page 216).

14.3.28 TERSE/VERBose

```
IBM VM/CMS and IBM PC/DOS
    TERSE
    VERB
DEC VAX/VMS
    /TERSE
    /VERB
```

Headings are not displayed on the terminal as the document is processed when the TERSE option is specified. Headings are displayed on the terminal as the document is processed when the VERBOSE option is specified. TERSE is the default when in line mode (see "LINEmode" on page 216).

14.3.29 TO

```
IBM VM/CMS and IBM PC/DOS
    TO page-number
DEC VAX/VMS
    /TO=page-number
```

The TO option will direct WATCOM Script/GML to stop printing the document at the specified page number within the body of the document. The number is specified as an integer, and does not depend on the format the numbers on the output pages (or whether or not they are even printed on the page). For example, if the page numbers are formatted as roman numerals, the page number you specify would be "6", not "vi". See "FROM" on page 215 for more information.

14.3.30 VALUESet

```
IBM VM/CMS and IBM PC/DOS
    VALUES file-name
DEC VAX/VMS
    /VALUES=file-name
```

The VALUESET option is an equivalent name for the MAILMERGE option. See "MAILmerge" on page 217 for more information.

14.3.31 WAIT/NOWAIT

```
IBM VM/CMS and IBM PC/DOS
    WAIT
    NOWAIT
DEC VAX/VMS
    /WAIT
    /NOWAIT
```

Certain errors (such as device not ready) will result in a query from WATCOM Script/GML about continuing with the document processing. WATCOM Script/GML also waits after processing the document when in not line mode (see "LINEmode" on page 216). The default option WAIT enables these queries. The option NOWAIT will suppress the query.

14.3.32 WARNing/NOWARNing

```
IBM VM/CMS and IBM PC/DOS
    WARN
    NOWARN
DEC VAX/VMS
    /WARN
    /NOWARN
```

The WARNING option(the default) causes GML warning messages about possible error conditions to be displayed on the screen. Processing of the document is not halted when a warning message is displayed. WATCOM Script/GML warnings about possible error situations and information messages are not displayed on the screen when the NOWARNING option is specified.

14.3.33 WSCRipt

```
IBM VM/CMS and IBM PC/DOS
    WSCR
DEC VAX/VMS
    /WSCR
```

The WSCRIPT option enables recognition of Script control words and the line separator character. In addition, it enables several WATCOM extensions over Waterloo Script. The extensions are:

1. Lines of input which

   - are processed when concatenate is OFF
   - start with blank space
   - the blank space is followed by a GML tag that is not a continuation

   have the blank space at the beginning of the line ignored.

2. When .CO OFF is set, lines which exceed the line length are split into two lines.

3. Extra blanks between words are suppressed in concatenate mode.

4. Full and partial stops are recognized anywhere in the input line if followed by a space.

5. If a macro for .LB, .LT, .NL, or .BL is not defined, a break is not implicitly performed.

Device Reference

15 Devices

15.1 Output Devices in WATCOM Script/GML

When you process a document, WATCOM Script/GML must create the resulting output for a particular output device. The format of the output may be different among devices, or for a device which can be set with different characteristics. To provide support for these differences, WATCOM Script/GML specifies a device as having several components.

Special characters, or **control sequences**, are sent to a device to perform various functions. For example, a control sequence is needed when a new output line or a new page is started. The definition of the control sequences required by WATCOM Script/GML for a particular device is called a **driver**.

The character sets, or **fonts**, that are available for the document is another component in the definition of a device. Each font definition specifies information such as the size of the characters. Some fonts have different sized characters to produce more attractive output.

Some of the differences in the format of the output for a device are related to the way in which the device is set up. The number of lines on a page, continuous forms or single sheet feeding, and default fonts are examples of set up differences. The combination of the font and driver definitions with the specification of the set up values create a **device** definition.

15.2 Page Addressing

A particular point on the output page is identified by a horizontal (X-axis) and a vertical (Y-axis) component. Together, the X and Y components designate the **address** of a point on the page. As each word and line of output is processed, the X and Y components of the address are adjusted to make a new address. Many devices restrict the adjustment of the address. Other devices are known as **point addressable** or **full page addressing** devices, and allow any point on the page to be addressed.

WATCOM Script/GML assumes that the start of an output page is the upper left corner. The horizontal component of the page address is adjusted for each character placed on

the output page. The vertical component of the page address is adjusted for each output line. The current X and Y address component values are available through the %X_ADDRESS and %Y_ADDRESS device functions. (See "X_ADDRESS" and "Y_ADDRESS" on page 240 for more information).

15.3 Augmented Device Definitions

Certain device operations are not selectable through the device and driver definitions. WATCOM Script/GML augments some device definitions by directly supporting these operations.

The augmented device definitions are recognized by the starting characters of the driver name. For example, **HPLDRV** is recognized as the name of a driver definition for the HP LaserJet printer.

Name Prefix	*Augmented Device*
HPL	HP LaserJet
HPLP	HP LaserJet Plus
MLT	Multiwriter V (emulation mode)
MLTE	Multiwriter V (express mode)
PCG	IBM PC Graphics
PS	PostScript

If the driver definition name begins with HPL (but not HPLP), the value returned by the %X_ADDRESS and %TABWIDTH device functions is in terms of decipoints instead of dots.

15.4 Creating a Definition

The WATCOM GENDEV (**GEN**erate **DEV**ice) program (see "Running WATCOM GENDEV" on page 277) creates a definition for use by WATCOM Script/GML. The font, driver and device definitions are each created separately. The font and driver definitions appropriate for a particular device are selected when the device definition is created.

A definition is first specified with a text editor. The resulting data file is then processed by the WATCOM GENDEV program, which produces a new file that is used by WATCOM Script/GML when processing a document. The definition files produced by the WATCOM GENDEV program are collected together in a definition library. This library will contain all of the definitions that may be required for the production of a

document. Each definition in the library is referred to as a **member** of the library. Refer to "Libraries" on page 293 for more information.

Many computer systems limit the size of a library member name. To minimize this restriction, every definition has two names associated with it. The **member name** is the name of the library member which contains the definition. The **defined name** is the name used by WATCOM Script/GML and WATCOM GENDEV when referring to the definition.

When a defined name is referenced, the member name associated with that defined name must be known. This is accomplished through the use of a "directory" file which contains the defined name and the associated member name for each definition in the library. This file is named **WGMLST**, and is automatically created when the WATCOM GENDEV program is used to process a definition. The name WGMLST must not be used as a member name for any of the definitions.

If the file 'device1' contains a definition, the following example shows how the definition can be generated.

```
GENDEV device1
```
Figure 77. *Generating a Definition*

For more information on running the GENDEV program, refer to "Running WATCOM GENDEV" on page 277.

15.5 Deleting a Definition

Specifying the **:delete** tag in the input for the GENDEV program will delete a definition. The WGMLST directory file is re-created with the entry for the specified definition name removed. The newly created WGMLST file must be updated to the definition library, and the definition member associated with the specified definition name removed.

```
:DELETE
   defined_name = 'character string'.
```
Figure 78. *Deleting a Definition*

```
:DELETE
   defined_name = 'epson'.
```
Figure 79. *Example of a Definition Deletion*

15.6 General Device Tags

15.6.1 CMT

```
Format: :CMT.
```

The information following the comment tag on the input line is treated as a comment. Text data and device tags in the input line following the comment tag are not processed. The comment tag must be placed at the beginning of each input record that is to be ignored. This tag may appear at any point in the device definition source, although it may not be placed between device tag attributes.

15.6.2 INCLUDE

```
Format: :INCLUDE file='file name'.
```

The value of the required attribute **file** is used as the name of the file to include. The content of the included file is processed by WATCOM GENDEV as if the data was in the original file. This tag provides the means whereby a definition may be specified using a collection of separate files. More than one definition may be included into one file for processing by WATCOM GENDEV.

When working on a PC/DOS system, the DOS environment symbol **GMLINC** may be set with an include file list. This symbol is defined in the same way as a library definition list (see "Defining a Library List" on page 297), and provides a list of alternate directories for file inclusion. If an included file is not defined in the current directory, the directories specified by the include path list are searched for the file. If the file is still not found, the directories specified by the DOS environment symbol **PATH** are searched.

15.7 Device Functions

When creating a device or driver definition, it may be necessary to enter non-printable characters or information which will not be available until the document is processed. Arithmetic operations may also have to be performed on the data while the document is being processed. Device functions allow you to specify this information in the device or driver definitions.

Most of the device functions operate on supplied parameter values and return either a **numeric** or **character** result.

A **numeric** value is a sequence of digits, or the result of a device function which returns a number. Numeric values may be in either decimal or hexadecimal form. Hexadecimal numeric values begin with a dollar($) sign and are composed of digits and the characters **A** through **F**.

A **character** value is a sequence of characters enclosed in either single(') or double(") quotation marks. The quotation marks surrounding the text are not part of the character value. If a quotation mark of the same type used to delimit the character value is to be part of the character text, it may be entered by specifying the quote character twice. Only one quote character will appear in the resulting character value. This should only be done when the quote character you wish to enter as part of the character text is the same quote character being used to delimit the character value. The following lines illustrate valid character values:

```
'hello 12'
"hello 12"
"he'llo 12"
"he""llo 12"
```

The following lines illustrate invalid character values:

```
hello 12
hello 12"
"he"llo 12"
```

The following line is a valid character value, but is probably not the correct specification.

```
"it''s 12"
```

The two single quotes will be part of the character value because double quotes are used to enclose the value.

The result of some device functions will be used as **final** values for the sequence being defined. A final value is sent directly to the output device. Some of the device functions produce results which are not suitable for use as a final value. The result of this type of function must be supplied as a parameter value to a device function which can produce a final value.

Prior to transmitting the device function sequences to the output device, WATCOM Script/GML translates each character of the sequence into another character. The translation values are defined in the font definitions used with the device. Some of the device functions produce final values which will not be translated.

Each device function name begins with the percent character(%) and is immediately followed by a left parenthesis. If the device function has any parameter values, the

value(s) follow the left parenthesis and are separated by commas. The device function is terminated with a right parenthesis.

15.7.1 ADD

```
%ADD(123,456)
```

The two required parameters must both be numeric. The sum of the two parameters is returned as a numeric result. The result of this device function may not be used as a final value.

15.7.2 BINARY1

```
%BINARY1(123)
```

The required parameter must be numeric. The result of this function is a one byte binary number ranging from 0 to 255 inclusive. The result of this device function is a final value, and may not be used as a parameter of another device function. The result is not translated when sent to the output device.

15.7.3 BINARY2

```
%BINARY2(1234)
```

The required parameter must be numeric. The result of this function is a two byte binary number ranging from 0 to 65535 inclusive. The result of this device function is a final value, and may not be used as a parameter of another device function. The result is not translated when sent to the output device.

15.7.4 BINARY4

```
%BINARY4(1234)
```

The required parameter must be numeric. The result of this function is a four byte binary number ranging from 0 to 4294967295 inclusive. On some machines, the largest integer value is a two byte binary number. If such a machine is being used, two of the four bytes will always be zero. The result of this device function is a final value, and may not be used as a parameter of another device function. The result is not translated when sent to the output device.

15.7.5 CANCEL

```
%CANCEL("bold")
```

Some devices may cancel more than one operation with a single control sequence. For example, some devices may stop underlining when bolding is turned off. The **cancel** device function specifies the type of device operation that has been cancelled. WATCOM Script/GML will then re-establish the cancelled operation. The possible values of the required character parameter are *bold*, *underline*, and the name of a font switch method (see "TYPE Attribute" on page 257). The name of a font switch method is specified when the device automatically switches to a default font. This device function may not be used as a parameter of another device function.

15.7.6 CLEARPC

```
%CLEARPC()
```

This device function causes WATCOM Script/GML to clear the screen of an IBM PC. It is used in the device definition, primarily with the pausing section. There is no effect when this function is used in a driver definition. There are no parameters to this device function, and it may not be used as a parameter of another device function.

15.7.7 CLEAR3270

```
%CLEAR3270()
```

This device function causes WATCOM Script/GML to clear the screen of an IBM 3270 type of terminal. It is used in the device definition, primarily with the pausing section. There is no effect when this function is used in a driver definition. There are no parameters to this device function, and it may not be used as a parameter of another device function.

15.7.8 DATE

```
%DATE()
```

The result of this device function is a character value representing the current date. If the symbol *&date* is defined, the value of this symbol is returned. There are no parameters to this device function, and the result may not be used as a final value.

15.7.9 DECIMAL

```
%DECIMAL(123)
```

The required parameter must be numeric. The result of this device function is a character value representing the given number. The result may not be used as a final value.

15.7.10 DEFAULT_WIDTH

```
%DEFAULT_WIDTH()
```

The result of this device function is a numeric value which represents the default width of a character in the current font. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.11 DIVIDE

```
%DIVIDE(124,12)
```

The two required parameters must both be numeric. The dividend from the integer division of the first parameter by the second parameter is returned as a numeric value. The remainder resulting from the division is not returned. The result of this device function may not be used as a final value.

15.7.12 FLUSHPAGE

```
%FLUSHPAGE()
```

This device function causes WATCOM Script/GML to flush the current page to the output device. The page flush is obtained by printing enough blank lines to fill the current page. If the size of the document page is greater than the size of the output device page, the page flush will print enough blank lines to flush the current device page. If no data has been output to the device, and the page is the first page in the document, the current page will not be flushed. There are no parameters to this device function, and it may not be used as a parameter of another device function.

15.7.13 FONT_HEIGHT

```
%FONT_HEIGHT()
```

The result of this device function is a numeric value which represents the height of the current font. Adding this value to the *%font_space* value gives the total height of the line. The return value is an integer value in one hundredths of a point. For example, 12.25 points is returned as the number 1225. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.14 FONT_SPACE

```
%FONT_SPACE()
```

The result of this device function is a numeric value which represents the space between lines in the current font. Adding this value to the *%font_height* value gives the total height of the line. The return value is an integer value in one hundredths of a point. For example, 12.25 points is returned as the number 1225. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.15 FONT_NUMBER

```
%FONT_NUMBER()
```

The result of this device function is a numeric value which represents the number of the current font. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.16 FONT_OUTNAME1

```
%FONT_OUTNAME1()
```

The result of this device function is a character value which represents the *outname1* value of the current font. The *outname1* value is specified in each font definition. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.17 FONT_OUTNAME2

```
%FONT_OUTNAME2()
```

The result of this device function is a character value which represents the *outname2* value of the current font. The *outname2* value is specified in each font definition. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.18 FONT_RESIDENT

```
%FONT_RESIDENT()
```

The result of this device function is a character value. The result value represents the resident status of the current font. The resident status for each font is specified in the *:devicefont* block of the device definition. When the font changes in the document, the value returned by this function will change accordingly. The value 'Y' will be returned if the font is resident in the device, while the value 'N' will be returned if it is not resident. There are no parameters to this device function, and the result may not be used as a final value.

15.7.19 HEX

```
%HEX(123)
```

The required parameter must be numeric. The result of this device function is a character value representing the given number in hexadecimal form. For example, the number value 255 would be returned as the character value 'FF'. Note that a dollar sign is not returned as part of the value. The result may not be used as a final value.

15.7.20 IMAGE

```
%IMAGE("hello")
```

The required parameter must be a character value. The result of this device function is a character value representing the given parameter, and is a final value. The result of this device function may not be used as a parameter of another device function, and is not translated when sent to the output device.

15.7.21 LINE_HEIGHT

```
%LINE_HEIGHT()
```

The result of this device function is a numeric value which represents the height of the current font. Adding this value to the *%line_space* value gives the total height of the line. The return value is in terms of the device vertical base units. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.22 LINE_SPACE

```
%LINE_SPACE()
```

The result of this device function is a numeric value which represents the space between lines in of the current font. Adding this value to the *%line_height* value gives the total

height of the line. The return value is in terms of the device vertical base units. When the font changes in the document, the value returned by this function will change accordingly. There are no parameters to this device function, and the result may not be used as a final value.

15.7.23 PAGES

```
%PAGES()
```

The result of this device function is a numeric value which represents the number of the current page being output. This number is not related to the numbering of pages in the formatted output. There are no parameters to this device function, and the result may not be used as a final value.

15.7.24 PAGE_DEPTH

```
%PAGE_DEPTH()
```

The result of this device function is a numeric value which represents the depth of the output page as defined in the device definition. There are no parameters to this device function, and the result may not be used as a final value.

15.7.25 PAGE_WIDTH

```
%PAGE_WIDTH()
```

The result of this device function is a numeric value which represents the width of the output page as defined in the device definition. There are no parameters to this device function, and the result may not be used as a final value.

15.7.26 RECORDBREAK

```
%RECORDBREAK()
```

WATCOM Script/GML forms a line of output for a device. With some devices, it is desirable to send several of these output lines together as one record. With other devices, each line and even some control sequences must be sent as separate records. WATCOM Script/GML assumes that each record may contain several output lines. The device function **RECORDBREAK** instructs WATCOM Script/GML to send the information in the current record to the output device. There are no parameters to this device function, and it may not be used as a parameter of another device function.

15.7.27 REMAINDER

```
%REMAINDER(124,12)
```

The two required parameters must both be numeric. The remainder from the division of the first parameter by the second parameter is returned as a numeric value. The result of this device function may not be used as a final value.

15.7.28 SLEEP

```
%SLEEP(30)
```

There is no result returned from this device function. The required parameter must be a non-negative integer number. Other device functions are not allowed as parameters to this device function. This device function causes WATCOM Script/GML to suspend document processing for the specified number of seconds. This device function may not be used as a parameter of another device function.

15.7.29 SUBTRACT

```
%SUBTRACT(456,123)
```

The two required parameters must both be numeric. The difference obtained by subtracting the second parameter from the first parameter is returned as a numeric value. The result of this device function may not be used as a final value.

15.7.30 TAB_WIDTH

```
%TAB_WIDTH()
```

When WATCOM Script/GML uses tabbing to produce white space in a horizontal direction, the result of this device function is a numeric value which represents the amount of space that is being tabbed over. There are no parameters to this device function, and the result may not be used as a final value.

15.7.31 TEXT

```
%TEXT("hello")
```

The required parameter must be a character value. The result of this device function is a character value representing the given parameter. The result of this device function is a final value, and may not be used as a parameter of another device function. The result is translated when sent to the output device.

15.7.32 THICKNESS

```
%THICKNESS()
```

The result of this device function is a numeric value which represents the current thickness of the rule line being drawn. The value returned by the function is set when drawing horizontal or vertical lines, and when drawing a box. There are no parameters to this device function, and the result may not be used as a final value.

15.7.33 TIME

```
%TIME()
```

The result of this device function is a character value representing the current time of day. If the symbol *&time* is defined, the value of this symbol is returned. There are no parameters to this device function, and the result may not be used as a final value.

15.7.34 WAIT

```
%WAIT()
```

This device function causes WATCOM Script/GML to suspend document processing until the enter key on the keyboard is depressed. This device function is used in the device definition, primarily with the pausing section. There is no effect when this function is used in a driver definition. There are no parameters to this device function, and it may not be used as a parameter of another device function.

15.7.35 WGML_HEADER

```
%WGML_HEADER()
```

The result of this device function is a character value which represents the WATCOM Script/GML header. The header is the character value which identifies the WATCOM Script/GML product and version number. There are no parameters to this device function, and the result may not be used as a final value.

15.7.36 X_ADDRESS

```
%X_ADDRESS()
```

The result of this device function is a numeric value which represents the current horizontal(X-axis) position on the output page. There are no parameters to this device

function, and the result may not be used as a final value. (See "Page Addressing" on page 227 for more information).

15.7.37 X_SIZE

```
%X_SIZE()
```

The result of this device function is a numeric value which represents the current horizontal length of a line to be drawn. The value of this function is set when drawing a box or a horizontal line. There are no parameters to this device function, and the result may not be used as a final value.

15.7.38 Y_ADDRESS

```
%Y_ADDRESS()
```

The result of this device function is a numeric value which represents the current vertical(Y-axis) position on the output page. There are no parameters to this device function, and the result may not be used as a final value. (See "Page Addressing" on page 227 for more information).

15.7.39 Y_SIZE

```
%Y_SIZE()
```

The result of this device function is a numeric value which represents the current vertical length of a line to be drawn. The value of this function is set when drawing a box or a horizontal line. There are no parameters to this device function, and the result may not be used as a final value.

15.8 Defining a Font

Information about the character sets available for a particular device is needed by WATCOM Script/GML to properly process a document. The **FONT** block is processed by the GENDEV program to create a font definition. The resulting definition is referenced on the WATCOM Script/GML command line and by the device definitions.

```
:FONT
   <attributes>
   <width block>
   <intrans block>
   <outtrans block>
:eFONT.
```

Figure 80. *The FONT Block*

A font block begins with the **:font** tag and ends with the **:efont** tag. The attributes of the font block must all be specified.

The width, intrans and outtrans character definition blocks are used to specify information which defines the characters of the font. Each possible character in a font is represented by a number between 0 and 255 inclusive (the representation depends on the machine system being used). Information about each of these characters may be specified in the character definition blocks, with one character definition per input line. If a character definition line is not specified for a particular character value, default values are supplied by the WATCOM GENDEV program.

Each of the character definition blocks are optional, and may be specified more that once and in any order. If specified, they must follow the font attributes.

15.8.1 Attributes of the Font Block

```
defined_name     = 'character string'
member_name      = 'character string'
font_out_name1   = 'character string'
font_out_name2   = 'character string'
line_height      = number
line_space       = number
scale_basis      = number
scale_min        = number
scale_max        = number
char_width       = number
mono_space_width = YES | NO
```

Figure 81. *Attributes of the FONT Block*

```
defined_name     = 'times-roman'
member_name      = 'PSTR'
font_out_name1   = 'Times-Roman'
line_height      = 1000
line_space       = 113
scale_basis      = 72000
scale_min        = 1000
scale_max        = 72000
char_width       = 250
mono_space_width = no
```

Figure 82. *Example of the FONT Block Attributes*

15.8.1.1 DEFINED_NAME Attribute

The **defined_name** attribute specifies the defined name of the font. Any valid character string may be used as the defined name. The defined name must be unique among the defined names of the font, driver and device definitions. The defined name is used to identify a font on the WATCOM Script/GML command line and is referred to in a device definition.

15.8.1.2 MEMBER_NAME Attribute

The **member_name** attribute specifies the member name of the font definition. The value of the member name attribute must be a valid file name. The member name must be unique among the member names of the font, driver and device definitions. When the GENDEV program processes the font block, it places the font definition in a file with the specified member name as the file name. If the file extension part of the file name is not specified, the GENDEV program will supply a default extension. Refer to "Running WATCOM GENDEV" on page 277 for more information.

15.8.1.3 FONT_OUT_NAME Attributes

The optional attributes **font_out_name1** and **font_out_name2** specify additional naming information for the font. The attribute value must be a valid character string.

Some devices require the specification of the font names or font characteristics during the initialization sequence or when a switch to a different font is made within the document. The font_out_name attributes may be used to supply these names to WATCOM Script/GML. If these values are not needed with a device, the null string('') may be specified.

15.8.1.4 LINE_HEIGHT Attribute

The **line_height** attribute specifies the height of the characters that are in the font being defined. The attribute value is a positive integer number which represents the height of the characters in terms of the vertical base units specified in the device definition if the font is not scaled. If the font is scaled, the value is in terms of the scale basis specified by the *scale_basis* attribute. This value added to the line space value is the total amount of space from one line to the next.

15.8.1.5 LINE_SPACE Attribute

The **line_space** attribute specifies the amount of space between two lines of characters that are in the font being defined. The attribute value is a positive integer number which represents the line space in terms of the vertical base units specified in the device definition if the font is not scaled. If the font is scaled, the value is in terms of the scale basis specified by the *scale_basis* attribute. This value added to the line height value is the total amount of space from one line to the next.

15.8.1.6 SCALE_BASIS Attribute

The **scale_basis** attribute specifies the number of base units per inch for scale operations. This attribute is not specified when the font characters have a fixed size.

15.8.1.7 SCALE_MIN Attribute

The **scale_min** attribute specifies the minimum size a font may be scaled to. The attribute value is a number in terms of the *scale_basis* attribute. This attribute must be specified if the font characters are scaled.

15.8.1.8 SCALE_MAX Attribute

The **scale_max** attribute specifies the maximim size a font may be scaled to. The attribute value is a number in terms of the *scale_basis* attribute. This attribute must be specified if the font characters are scaled.

15.8.1.9 CHAR_WIDTH Attribute

The **char_width** attribute specifies the default width of the characters that are in the font being defined. The attribute value is a positive integer number which represents the width of a character in terms of the horizontal base units specified in the device definition if the font is not scaled. For example, if there are ten characters per inch with a particular font, and there are 300 horizontal base units per inch, then the number 30

would be the default character width value for the font. If the font is scaled, the value is in terms of the scale basis specified by the *scale_basis* attribute.

15.8.1.10 MONO_SPACE_WIDTH Attribute

The optional **mono_space_width** attribute determines the way in which the character widths of the font are used. The attribute value may be one of the keywords YES and NO.

YES — All of the characters in the font are treated as having the same width.

NO — The font is treated as a character set with varying widths.

15.8.2 Width Block

The **width** block is an optional section of the font block. The width block is specified within a font definition after the font block attributes.

```
:WIDTH.
font-character  character-width
:eWIDTH.
```

Figure 83. *The WIDTH Block*

The **:width** tag begins the width block. The **:ewidth** tag delimits the end of the width block and must be the first non-space characters on the line. Hexadecimal values in the width block must begin with a dollar($) sign.

```
:WIDTH.
1    0
A   20
$C2 20
:eWIDTH.
```

Figure 84. *Example of the WIDTH Block*

The font-character value specifies the character to define. This value may be a single character, an integer number, or a hexadecimal number. If a single character is specified, it should not be enclosed in quotes.

The character-width value specifies the width of the character in terms of the horizontal base units specified in the device definition if the font is fixed, and in terms of the scale basis if the font characters are scaled. This value must be a non-negative integer

number. If a character is not defined in the width block, it is assigned the width value defined by the char_width attribute of the font block.

15.8.3 InTrans Block

The **intrans** block is an optional section of the font block. The intrans block is specified within a font definition after the font block attributes.

```
:INTRANS.
font-character  input-translation
:eINTRANS.
```

Figure 85. *The INTRANS Block*

The **:intrans** tag begins the intrans block. The **:eintrans** tag delimits the end of the intrans block and must be the first non-space characters on the line.

```
:INTRANS.
1     1
A   $C1
$C2 $40
:eINTRANS.
```

Figure 86. *Example of the INTRANS Block*

The font-character and input-translation values may be a single character, an integer number, or a hexadecimal number. Hexadecimal values in the intrans block must begin with a dollar($) sign. If a single character is specified, it should not be enclosed in quotes.

The input-translation value defines the character that the font-character value is translated to when input translation is performed. If a character is not defined in the intrans block, the input translation value of the character will be itself. Refer to "Input Translation" on page 80 for more information.

15.8.4 OutTrans Block

The **outtrans** block is an optional section of the font block. The outtrans block is specified within a font definition after the font block attributes.

```
:OUTTRANS.
font-character  output-translation
:eOUTTRANS.
```
Figure 87. *The OUTTRANS Block*

The **:outtrans** tag begins the outtrans block. The **:eouttrans** tag delimits the end of the outtrans block and must be the first non-space characters on the line.

```
:OUTTRANS.
1     1
A   $C1
$C2 $40
)   \  )
175 \  2  6  7
:eOUTTRANS.
```
Figure 88. *Example of the OUTTRANS Block*

The font-character value specifies the character to define. This value may be a single character, an integer number, or a hexadecimal number. Hexadecimal values in the outtrans block must begin with a dollar($) sign. If a single character is specified, it should not be enclosed in quotes.

Some output devices represent characters by a different numeric sequence than the computer used to produce the document. The output translation value may be specified to translate the character entered in the input text to the numeric representation required by the device. Each output translation character is separated by a space, and may be a character, an integer number, or a hexadecimal number. If a character is not defined in the outtrans block, the output translation value of the character will be itself.

15.9 Defining a Driver

A driver definition specifies the control sequences and methods by which output is produced. This includes such things as how to do a font change, bolding, and underlining. The **DRIVER** block is processed by the GENDEV program to create a driver definition. The resulting definition is referenced by the device definitions.

```
:DRIVER
   <attributes>
   <init block>
   <finish block>
   <newline block>
   <newpage block>
   <htab block>
   <boldstart block>
   <boldend block>
   <understart block>
   <underend block>
   <fontswitch block>
   <pageaddress block>
   <absoluteaddress block>
   <hline block>
   <vline block>
   <dbox block>
:eDRIVER.
```

Figure 89. *The DRIVER Block*

A driver block begins with the **:driver** tag and ends with the **:edriver** tag. The attributes of the driver block must all be specified. The various blocks of information following the attributes are not all required to define the driver. When they are specified, they must be in the order shown in Figure 89.

15.9.1 Attributes of the Driver Block

```
defined_name     = 'character string'
member_name      = 'character string'
rec_spec         = 'character string'
fill_char        = number | character
```

Figure 90. *Attributes of the DRIVER Block*

```
defined_name     = 'x2700drv'
member_name      = 'x27drv'
rec_spec         = '(f:80)'
fill_char        = 0
```

Figure 91. *Example of the DRIVER Block Attributes*

15.9.1.1 DEFINED_NAME Attribute

The **defined_name** attribute specifies the defined name of the driver. Any valid character string may be used as the defined name. The defined name must be unique among the defined names of the font, driver and device definitions. The defined name is used to identify the driver when creating a device definition.

15.9.1.2 MEMBER_NAME Attribute

The **member_name** attribute specifies the member name of the driver definition. The value of the member name attribute must be a valid file name. The member name must be unique among the member names of the font, driver and device definitions. When the GENDEV program processes the driver block, it places the driver definition in a file with the specified member name as the file name. If the file extension part of the file name is not specified, the GENDEV program will supply a default extension. Refer to "Running WATCOM GENDEV" on page 277 for more information.

15.9.1.3 REC_SPEC Attribute

The **rec_spec** attribute specifies a record specification value (for example, either (f:80) or (f:c:80) are allowed) for the output device. The attribute value must be a valid record specification (see "Files" on page 281).

15.9.1.4 FILL_CHAR Attribute

A fill character is needed when the output records for the device have a fixed length. If a record is output which is less than the record length, the record must be filled out to the required length. The **fill_char** attribute specifies the fill character to be used when doing this record filling. The attribute value may be a single character value enclosed in quotes, an integer number, or a hexadecimal number. A hexadecimal number is preceded by the dollar($) sign.

15.9.2 INIT Block

The **init** block specifies the initialization values which are to be output to a device. If no initialization is required, the init block may be omitted. The two value sections of the init block may appear in any order and as many times as necessary.

```
:INIT
   place= START | DOCUMENT
   :value.
      <device functions>
   :evalue.
   :fontvalue.
      <device functions>
   :efontvalue.
:eINIT.
```

Figure 92. *The INIT Block*

The init block begins with the **:init** tag and ends with the **:einit** tag. The place attribute and at least one of the value or fontvalue sections must be specified.

```
:INIT
   place= start
   :fontvalue.
      %text(%font_outname2())%binary1(0)
      %text(%font_outname1())%recordbreak()
   :efontvalue.
   :value.
      %binary1($27)
      %text('+P,X2700 -- WATCOM Script/GML -- ')
      %binary1($15)%recordbreak()
   :evalue.
:eINIT.
```

Figure 93. *Example of the INIT Block*

15.9.2.1 PLACE Attribute

The **place** attribute indicates where in the output the specified initialization sequences are to appear. The attribute value may be the keyword *START* or *DOCUMENT*. A separate init block may be specified for each of these two values. An automatic font select of font zero is performed at the beginning of a document as part of the initialization.

START The init block is evaluated when WATCOM Script/GML starts processing the input source.

DOCUMENT The init block is evaluated when WATCOM Script/GML starts processing a document.

15.9.2.2 VALUE Section

The **value** section specifies the general initialization sequence to be output, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line. The value section may be specified more than once, and may precede and/or follow a fontvalue section.

15.9.2.3 FONTVALUE Section

The **fontvalue** section is used to perform font initialization, and is started with the **:fontvalue** tag. Device functions are then entered after the :fontvalue tag, and may be specified on more than one line. WATCOM Script/GML selects the fonts being used in the document. For each of the selected fonts, the fontvalue section is evaluated. Device functions, such %default_width, will return the values appropriate for the selected font. The **:efontvalue** tag delimits the end of a fontvalue section, and must be the first

non-space characters in the line. The fontvalue section may be specified more than once, and may precede and/or follow a value section.

15.9.3 FINISH Block

The **finish** block specifies the finalization values which are to be output. If no finalization is required, the finish block may be omitted.

```
:FINISH
   place= DOCUMENT | END
   :value.
      <device functions>
   :evalue.
:eFINISH.
```

Figure 94. *The FINISH Block*

The finish block begins with the **:finish** tag and ends with the **:efinish** tag. The place attribute and the value section must both be specified.

```
:FINISH
   place=end
   :value.
      %binary1(21)%binary1($27)%text('+X')
      %binary1(21)%recordbreak()
   :evalue.
:eFINISH.
```

Figure 95. *Example of the FINISH Block*

15.9.3.1 PLACE Attribute

The **place** attribute indicates where in the output the specified finalization sequence is to appear. The attribute value may be the word *DOCUMENT* or *END*. A separate finish block may be specified for each of these two values.

DOCUMENT The finish block is evaluated when WATCOM Script/GML finishes processing a document.

END The finish block is evaluated when WATCOM Script/GML finishes processing the input source.

15.9.3.2 VALUE Section

The **value** section specifies the general finalization sequence to be output, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.4 NEWLINE Block

When WATCOM Script/GML starts a new line in the output, it must know the method by which to start the new line. The **newline** block specifies this method. To provide for different spacing mechanisms, the newline block may be specified any number of times to set different sequences for advancing lines with different line spacing.

```
:NEWLINE
   advance=number
   :value.
      <device functions>
   :evalue.
:eNEWLINE.
```

Figure 96. *The NEWLINE Block*

The newline block begins with the **:newline** tag and ends with the **:enewline** tag. The advance attribute and the value section must both be specified.

```
:NEWLINE
   advance=1
   :value.
      %binary1(13)%binary1(10)
   :evalue.
:eNEWLINE.
```

Figure 97. *Example of the NEWLINE Block*

15.9.4.1 ADVANCE Attribute

The **advance** attribute specifies the number of lines that will be advanced when the given sequence is output. The attribute value must be a non-negative integer. A particular advance number may be used only once in a newline block. The value zero gives the return to beginning of line sequence. This must be supplied if underlining and/or bolding is accomplished with overstriking. A newline block with an advance value of one must be specified.

15.9.4.2 VALUE Section

The **value** section specifies the sequence to be output to obtain the required number of line advances, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.5 NEWPAGE Block

The **newpage** block defines the method by which WATCOM Script/GML will start a new page in the output, and must be specified.

```
:NEWPAGE
   :value.
      <device functions>
   :evalue.
:eNEWPAGE.
```

Figure 98. *The NEWPAGE Block*

The newpage block begins with the **:newpage** tag and ends with the **:enewpage** tag. The value section must be specified.

```
:NEWPAGE
   :value.
      %binary1(12)
   :evalue.
:eNEWPAGE.
```

Figure 99. *Example of the NEWPAGE Block*

15.9.5.1 VALUE Section

The **value** section specifies the sequence to be output to obtain the new page, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.6 HTAB Block

The **htab** block defines a relative horizontal tabbing sequence. It must define a method of moving in a forward horizontal direction based upon the horizontal base unit measurement of the device. This block is not required, but will allow the production of a more professional looking document. If a proportional font is used, and this block is

not specified, it may not be possible to properly align the words to the right hand side of the column when the text is to be justified.

```
:HTAB
   :value.
      <device functions>
   :evalue.
:eHTAB.
```

Figure 100. *The HTAB Block*

The htab block begins with the **:htab** tag and ends with the **:ehtab** tag. The value section must be specified.

```
:CMT.Set up for horizontal tabbing.
:CMT.Issue tab control codes.
:CMT.Add tab width value over 256 to the '@' character
:CMT.Divide the tab width value between 16 and 256
:CMT.by 16 and add to the '@' character.
:CMT.Add the tab width value between 0 and 16
:CMT.to the '@' character.
:HTAB
   :value.
      %binary1(27)%text('H')
      %binary1(%add(%divide(%tab_width(),256),$40))
      %binary1(%add(%divide(%remainder(
               %tab_width(),256),16),$40))
      %binary1(%add(%remainder(%tab_width(),16),$40))
   :evalue.
:eHTAB.
```

Figure 101. *Example of the HTAB Block*

15.9.6.1 VALUE Section

The **value** section specifies the sequence to be output to obtain the horizontal tab, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line. The device function TAB_WIDTH will contain the amount of horizontal space WATCOM Script/GML will needs to tab over.

15.9.7 BOLDSTART Block

The **boldstart** block defines the method by which WATCOM Script/GML will cause text to appear in boldface in the output. If this block is not specified, bold text is obtained by overprinting the output line. If bolding is accomplished with overprinting, a newline block with an advance of zero must be specified.

```
:BOLDSTART
   :value.
      <device functions>
   :evalue.
:eBOLDSTART.
```
Figure 102. *The BOLDSTART Block*

The boldstart block begins with the **:boldstart** tag and ends with the **:eboldstart** tag. The value section must be specified.

```
:BOLDSTART
   :value.
      %binary1($27)%text('b')
   :evalue.
:eBOLDSTART.
```
Figure 103. *Example of the BOLDSTART Block*

15.9.7.1 VALUE Section

The **value** section specifies the sequence to be output to obtain the bolding, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.8 BOLDEND Block

The **boldend** block defines the sequences needed to stop the bolding of text. The boldend block is required if a boldstart block is specified.

```
:BOLDEND
   :value.
      <device functions>
   :evalue.
:eBOLDEND.
```
Figure 104. *The BOLDEND Block*

The boldend block begins with the **:boldend** tag and ends with the **:eboldend** tag. The value section must be specified.

```
:BOLDEND
   :value.
      %binary1($27)%text('p')
   :evalue.
:eBOLDEND.
```
Figure 105. *Example of the BOLDEND Block*

15.9.8.1 VALUE Section

The **value** section specifies the sequence to be output to stop the bolding of text, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.9 UNDERSTART Block

The **understart** block defines the method by which WATCOM Script/GML will cause text to be underscored in the output. If this block is not specified, underscoring of text is obtained by overprinting the output line. If overprinting is used, a newline block with an advance of zero must be specified.

```
:UNDERSTART
   :value.
      <device functions>
   :evalue.
:eUNDERSTART.
```
Figure 106. *The UNDERSTART Block*

The understart block begins with the **:understart** tag and ends with the **:eunderstart** tag. The value section must be specified.

```
:UNDERSTART
   :value.
      %binary1($27)%text('u')
   :evalue.
:eUNDERSTART.
```
Figure 107. *Example of the UNDERSTART Block*

15.9.9.1 VALUE Section

The **value** section specifies the sequence to be output to obtain the underscoring, and is started with the **:value** tag. Device functions are then entered after the :value tag, and

may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.10 UNDEREND Block

The **underend** block defines the sequences needed to stop the underscoring of text. The underend block is required if an understart block is specified.

```
:UNDEREND
   :value.
      <device functions>
   :evalue.
:eUNDEREND.
```

Figure 108. *The UNDEREND Block*

The underend block begins with the **:underend** tag and ends with the **:eunderend** tag. The value section must be specified.

```
:UNDEREND
   :value.
      %binary1($27)%text('w')
   :evalue.
:eUNDEREND.
```

Figure 109. *Example of the UNDEREND Block*

15.9.10.1 VALUE Section

The **value** section specifies the sequence to be output to stop the underscoring of text, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.11 FONTSWITCH Block

The **fontswitch** block identifies a method for switching fonts. With some output devices, different fonts are available when control sequences are used to switch from the default font. A separate fontswitch block may be specified for each type of font switch method available with the device.

```
:FONTSWITCH
   type=string
   :startvalue.
      <device functions>
   :estartvalue.
   :endvalue.
      <device functions>
   :eendvalue.
:eFONTSWITCH.
```

Figure 110. *The FONTSWITCH Block*

The fontswitch block begins with the **:fontswitch** tag and ends with the **:efontswitch** tag. The type attribute and the value section must be specified.

```
:FONTSWITCH
   type='qume proportional on'
   :startvalue.
      %binary1(27)%text('$')
   :estartvalue.
   :endvalue.
      %binary1(27)%text('x')
      %recordbreak()
   :eendvalue.
:eFONTSWITCH.
```

Figure 111. *Example of the FONTSWITCH Block*

15.9.11.1 TYPE Attribute

The character value of the **type** attribute provides an identifier for the font switch method. This identifier is referenced in the device definition when specifying the fonts available for the device. The attribute value must be unique among the font switch blocks in the driver definition.

15.9.11.2 STARTVALUE Section

The **startvalue** section specifies the sequence to be output to perform the font switch, and is started with the **:startvalue** tag. Device functions are then entered after the :startvalue tag, and may be specified on more than one line. The **:estartvalue** tag delimits the end of a startvalue section, and must be the first non-space characters in the line.

When a switch between two fonts is necessary, the startvalue sections of the two fonts are evaluated. The font switch is only performed if the results of the two evaluations are different.

15.9.11.3 ENDVALUE Section

The **endvalue** section specifies the sequence to be output before the font switch sequence of the new font is performed, and is started with the **:endvalue** tag. Device functions are then entered after the :endvalue tag, and may be specified on more than one line. The **:eendvalue** tag delimits the end of an endvalue section, and must be the first non-space characters in the line.

15.9.12 PAGEADDRESS Block

As text is placed on the output page, the X and Y components of the address are adjusted to make a new address. With some output devices, this adjustment is added (positive) to the address. The adjustment is subtracted (negative) with other output devices. The **pageaddress** block specifies whether the adjustment is positive or negative. If the output device does not support page addressing, this block should not be specified. (See "Page Addressing" on page 227 for more information).

```
:PAGEADDRESS
   x_positive = YES | NO
   y_positive = YES | NO
:ePAGEADDRESS.
```

Figure 112. *The PAGEADDRESS Block*

The pageaddress block begins with the **:pageaddress** tag and ends with the **:epageaddress** tag. The two attributes must be specified.

```
:PAGEADDRESS
   x_positive = yes
   y_positive = yes
:ePAGEADDRESS.
```

Figure 113. *Example of the PAGEADDRESS Block*

15.9.13 ABSOLUTEADDRESS Block

When an output device supports page addressing, the **absoluteaddress** block specifies the mechanism for absolute page addressing. If the output device does not support page addressing, this block should not be specified.

```
:ABSOLUTEADDRESS
   :value.
      <device functions>
   :evalue.
:eABSOLUTEADDRESS.
```
Figure 114. *The ABSOLUTEADDRESS Block*

The absoluteaddress block begins with the **:absoluteaddress** tag and ends with the **:eabsoluteaddress** tag. The value section must be specified.

```
:ABSOLUTEADDRESS
   :value.
      %binary1($27)%text('a')
      %text(decimal(%x_address()))
      %text(',')%decimal(%y_address())
      %binary1($15)
   :evalue.
:eABSOLUTEADDRESS.
```
Figure 115. *Example of the ABSOLUTEADDRESS Block*

15.9.13.1 VALUE Section

The **value** section specifies the sequence to be output to set a new absolute address, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.14 HLINE Block

The **hline** block specifies the mechanism for creating horizontal rule lines. If this block is not specified, rule lines will be created with characters.

```
:HLINE
   thickness = number
   :value.
      <device functions>
   :evalue.
:eHLINE.
```
Figure 116. *The HLINE Block*

The hline block begins with the **:hline** tag and ends with the **:ehline** tag. The thickness attribute and value section must be specified. The special symbols *%x_size* and *%thickness* are defined prior to processing the hline block. The symbol %x_size is set

to the width of the horizontal line, from the left edge to the right edge. The symbol %thickness is set to the value specified by the hline blocks thickness attribute. WATCOM Script/GML positions to the bottom left corner of the line before creating the rule line, and assumes the current point of the device is set to the bottom right corner of the line when finished.

```
:HLINE
   thickness=4
   :value.
      %binary1(27)%text('x')
      %text(%decimal(%x_address()))%text(',')
      %text(%decimal(%y_address()))%text(',')
      %text(%decimal(%x_size()))%text(',')
      %text(%decimal(%thickness()))%text(',')%binary1(10)
   :evalue.
:eHLINE.
```

Figure 117. *Example of the HLINE Block*

15.9.14.1 THICKNESS Attribute

The **thickness** attribute specifies the thickness of the horizontal line. This value is in terms of the device horizontal base units.

15.9.14.2 VALUE Section

The **value** section specifies the sequence to be output to create a horizontal rule line, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.15 VLINE Block

The **vline** block specifies the mechanism for creating vertical rule lines. If this block is not specified, rule lines will be created with characters.

```
:VLINE
   thickness = number
   :value.
      <device functions>
   :evalue.
:eVLINE.
```

Figure 118. *The VLINE Block*

The vline block begins with the **:vline** tag and ends with the **:evline** tag. The thickness attribute and value section must be specified. The special symbols *%y_size* and *%thickness* are defined prior to processing the vline block. The symbol %y_size is set to the height of the vertical line, from the top edge to the bottom edge. The symbol %thickness is set to the value specified by the vline blocks thickness attribute. WATCOM Script/GML positions to the bottom left corner of the line before creating the rule line, and assumes the current point of the device is set to the top left corner of the line when finished.

```
:VLINE
   thickness=4
   :value.
      %binary1(27)%text('y')
      %text(%decimal(%x_address()))%text(',')
      %text(%decimal(%y_address()))%text(',')
      %text(%decimal(%y_size()))%text(',')
      %text(%decimal(%thickness()))%text(',')%binary1(10)
   :evalue.
:eVLINE.
```

Figure 119. *Example of the VLINE Block*

15.9.15.1 THICKNESS Attribute

The **thickness** attribute specifies the thickness of the vertical line. This value is in terms of the device horizontal base units.

15.9.15.2 VALUE Section

The **value** section specifies the sequence to be output to create a vertical rule line, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.9.16 DBOX Block

The **dbox** block specifies the mechanism for creating a box. If this block is not specified, rule lines will be created with the hline and vline block definitions.

```
:DBOX
   thickness = number
   :value.
      <device functions>
   :evalue.
:eDBOX.
```

Figure 120. *The DBOX Block*

The dbox block begins with the **:dbox** tag and ends with the **:edbox** tag. The thickness attribute and value section must be specified. The special symbols *%x_size*, *%y_size* and *%thickness* are defined prior to processing the dbox block. The symbol %x_size is set to the width of the horizontal component of the box, from the left edge to the right edge. The symbol %y_size is set to the height of the vertical component of the box, from the top edge to the bottom edge. The symbol %thickness is set to the value specified by the dbox thickness attribute. WATCOM Script/GML positions to the bottom left corner of the box before creating the box lines, and assumes the current point of the device is set to the bottom right corner of the box when finished.

```
:DBOX
   thickness=10
   :value.
      %recordbreak()
      %text(%decimal(%divide(%thickness(),2)))
      %text(' ')
      %text(%decimal(%divide(%thickness(),2)))
      %text(' rmoveto')
      %recordbreak()
      %text('0 ')%text(%decimal(%y_size()))%text(' rlineto ')
      %recordbreak()
      %text(%decimal(%subtract(%x_size(),%thickness())))
      %text(' 0 rlineto ')
      %recordbreak()
      %text('0 -')%text(%decimal(%y_size()))%text(' rlineto ')
      %recordbreak()
      %text('-')%text(%decimal(%subtract(%x_size(),%thickness())))
      %text(' 0 rlineto')
      %recordbreak()
      %text('closepath ')
      %text(%decimal(%thickness()))
      %text(' setlinewidth stroke')
      %recordbreak()
      %text(%decimal(%add(%x_address(),%x_size())))
      %text(' ')%text(%decimal(%y_address()))
      %text(' moveto')
      %recordbreak()
   :evalue.
:eDBOX.
```

Figure 121. *Example of the DBOX Block*

15.9.16.1 THICKNESS Attribute

The **thickness** attribute specifies the thickness of the box lines. This value is in terms of the device horizontal base units.

15.9.16.2 VALUE Section

The **value** section specifies the sequence to be output to create a box, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.10 Defining a Device

The device definition combines the specification of device setup characteristics, driver definition, and font definitions. The **DEVICE** block is processed by the GENDEV program to create a device definition. The device definition to be used for the processing of a document is specified on the WATCOM Script/GML command line.

```
:DEVICE
   <attributes>
   <pause block>
   <devicefont block>
   <defaultfont block>
   <fontpause block>
   <rule block>
   <box block>
   <underscore block>
   <pagestart block>
:eDEVICE.
```

Figure 122. *The DEVICE Block*

A device block begins with the **:device** tag and ends with the **:edevice** tag. The attributes of the device block must all be specified. The various blocks of information following the attributes are not all required to define the device. When they are specified, they must be in the order shown in Figure 122.

15.10.1 Attributes of the Device Block

```
defined_name          = 'character string'
member_name           = 'character string'
driver_name           = 'character string'
output_name           = 'character string'
output_suffix         = 'character string'
page_width            = horizontal-base-units
page_depth            = vertical-base-units
horizontal_base_units = number
vertical_base_units   = number
```

Figure 123. *Device Attributes*

```
defined_name          = 'x2700'
member_name           = 'x2700'
driver_name           = 'x2700drv'
output_name           = ''
output_suffix         = 'x2700'
page_width            = 2400
page_depth            = 3300
horizontal_base_units = 300
vertical_base_units   = 300
```

Figure 124. *Example of the Device Attributes*

15.10.1.1 DEFINED_NAME Attribute

The **defined_name** attribute specifies the defined name of the device. Any valid character string may be used as the defined name. The defined name must be unique among the defined names of the font, driver and device definitions. The defined name is used to identify the device on the WATCOM Script/GML command line.

15.10.1.2 MEMBER_NAME Attribute

The **member_name** attribute specifies the member name of the device definition. The value of the member name attribute must be a valid file name. The member name must be unique among the member names of the font, driver and device definitions. When the GENDEV program processes the device block, it places the device definition in a file with the specified member name as the file name. If the file extension part of the file name is not specified, the GENDEV program will supply a default extension. Refer to "Running WATCOM GENDEV" on page 277 for more information.

15.10.1.3 DRIVER_NAME Attribute

The **driver_name** attribute specifies a character value that is the defined name of a driver definition. The driver definition contains the control sequences used to produce the output for the device.

15.10.1.4 OUTPUT_NAME Attribute

If the output is to be directed to a specific device, such as a printer, the **output_name** attribute specifies the name of that device. A record specification may be part of the output name (see "Files" on page 281). If a device does not exist with the specified name, the attribute value will be used to create an output file name. If the null string('') is specified as the attribute value, the file name of the input source document will be used.

15.10.1.5 OUTPUT_SUFFIX Attribute

The character value of the **output_suffix** attribute will be suffixed to the output name. If the null string('') is specified as the attribute value, no file suffix will be added.

15.10.1.6 PAGE_WIDTH Attribute

The **page_width** attribute specifies the physical page width of the output page in horizontal base units. The page width defined in the document layout may be smaller than this value.

15.10.1.7 PAGE_DEPTH Attribute

The **page_depth** attribute specifies the physical page depth of the output page in vertical base units. The page depth defined in the document layout may be smaller or larger than this value. If the page depth in the document layout is larger than this value, WATCOM Script/GML will produce one document page over several of the device pages.

15.10.1.8 HORIZONTAL_BASE_UNITS Attribute

The value of the **horizontal_base_units** attribute is a positive integer number. A horizontal base unit is the smallest unit of space that the output device can advance in a horizontal direction. The attribute value specifies the number of horizontal base units which are equivalent to one inch of horizontal space.

15.10.1.9 VERTICAL_BASE_UNITS Attribute

The value of the **vertical_base_units** attribute is a positive integer number. A vertical base unit is the smallest unit of space that the output device can advance in a vertical direction. The attribute value specifies the number of vertical base units which are equivalent to one inch of vertical space.

15.10.2 PAUSE Block

The **pause** block is used to cause various actions to occur at the terminal while WATCOM Script/GML is processing the document. The issuing of messages and pausing while changes to the output device are made can be controlled through the pause block. If pausing or messages are not required, the pause block may be omitted.

```
:PAUSE
   place                = START|DOCUMENT
                          |DOCUMENT_PAGE
                          |DEVICE_PAGE
   :value.
      <device functions>
   :evalue.
:ePAUSE.
```

Figure 125. *The PAUSE Block*

A pause block begins with the **:pause** tag and ends with the **:epause** tag. The place attribute and value section must both be specified.

```
:PAUSE
   place = document
   :value.
      %text( "Press enter to start the document." )
      %recordbreak()%wait()%clear3270()
   :evalue.
:ePAUSE.
```

Figure 126. *Example of the PAUSE Block*

15.10.2.1 PLACE Attribute

The **place** attribute specifies the point during processing when WATCOM Script/GML should evaluate the pause block. The attribute value may be one of the keywords *START*, *DOCUMENT*, *DOCUMENT_PAGE*, or *DEVICE_PAGE*. A separate pause block may be specified for each of the different attribute values.

START The pause block is evaluated when WATCOM Script/GML begins processing the source input data.

DOCUMENT
The pause block is evaluated when WATCOM Script/GML begins processing the document text.

DOCUMENT_PAGE
The pause block is evaluated at the beginning of each document page. A document page is the amount of output that WATCOM Script/GML formats for a page in the document. The document page may be smaller or larger than the physical page produced by the output device. If the page being printed is both the document page and the device page, the document page pause block takes precedence over the device page pause block.

DEVICE_PAGE
The pause block is evaluated when WATCOM Script/GML begins a new page on the output device.

15.10.2.2 VALUE Section

The **value** section specifies the pausing control sequences to be output, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.10.3 DEVICEFONT Block

A number of different device definitions may use the same fonts. In some cases, the fonts available to one device may be a subset of the fonts available to another device. The device font block is used to define the fonts available to the device definition. One device font block is specified for each of the available fonts.

```
:DEVICEFONT
   fontname    = 'character string'
   fontswitch  = 'character string'
   fontpause   = 'character string'
   resident    = YES | NO
:eDEVICEFONT.
```

Figure 127. *The DEVICEFONT Block*

A devicefont block begins with the **:devicefont** tag and ends with the **:edevicefont** tag. All of the attributes of a device font block must be specified.

```
:DEVICEFONT
        fontname   = 'vintage12'
        fontswitch = "x27 font switch"
        fontpause  = ""
        resident   = no
:eDEVICEFONT.
```

Figure 128. *Example of the DEVICEFONT Block*

15.10.3.1 FONTNAME Attribute

The **fontname** attribute specifies a character value. This value is the defined name value in a font definition.

15.10.3.2 FONTSWITCH Attribute

The **fontswitch** attribute specifies a character value which is the font switch method to be used when switching into the font. The attribute value must be an identifier defined by a font switch block in the driver definition. If the value is the null('') string, no font switching is performed.

15.10.3.3 FONTPAUSE Attribute

The **fontpause** attribute specifies a character value which is the font pausing method to be used when switching into the font. The attribute value must be an identifier defined by a font pause block in the current device definition. If the value is the null('') string, no font pausing is performed.

15.10.3.4 RESIDENT Attribute

The **resident** attribute defines the resident status of the font in the output device. With some devices, information must be sent to the device for each of the fonts being used. To reduce the time required for printing a document, information about the most commonly used fonts often 'reside' on the device permanently. The keywords *YES* or *NO* may be specified as the attribute value.

YES	The font resides permanently on the device.
NO	The font does not reside on the device.

15.10.4 DEFAULTFONT Block

The **defaultfont** block specifies a font to be used with this device when processing a document. A default font block is specified for each default font to be defined.

```
:DEFAULTFONT
   font                = number
   fontname            = 'character string'
   font_height         = number
   font_space          = number
   fontstyle           = PLAIN|UNDERSCORE
                         |BOLD|USBOLD
                         |UNDERLINE|ULBOLD
:eDEFAULTFONT.
```

Figure 129. *The DEFAULTFONT Block*

A default font block begins with the **:defaultfont** tag and ends with the **:edefaultfont** tag. Most of the attributes in the default font block must be specified. The attributes font_height and font_space are only necessary if the font is scaled.

```
:DEFAULTFONT
   font         = 1
   fontname     = 'times-italic'
   font_height  = 10
   font_space   = 2
   fontstyle    = plain
:eDEFAULTFONT.
```

Figure 130. *Example of the DEFAULTFONT Block*

15.10.4.1 FONT Attribute

The value of the **font** attribute must be a non-negative integer number value. The font numbers zero through three correspond to the highlight phrase tags in the GML document. They also correspond directly to the font numbers used in the layout definition. The default font with a font number of zero must be specified. All font numbers used in the document which are not defined with the default font block or the font option on the WATCOM Script/GML command line will be assigned the values specified in the font zero default font block.

15.10.4.2 FONTNAME Attribute

The **fontname** attribute specifies a character value. This value is a font name defined by a device font block in the device definition. The font name must be specified in a devicefont block within the current device definition.

15.10.4.3 FONT_HEIGHT Attribute

The **font_height** attribute specifies a number value. This value is the point size of the characters in a scaled font. The attribute does not have to be specified if the font is not scaled.

15.10.4.4 FONT_SPACE Attribute

The **font_space** attribute specifies a number value. This value is the space between lines of the font, and is specified in points. The attribute does not have to be specified if the font is not scaled.

15.10.4.5 FONTSTYLE Attribute

The font style values may be applied to any font. The value of the **fontstyle** attribute value is a keyword value which defines the style of the font. The keyword value may be one of *PLAIN*, *UNDERSCORE*, *BOLD*, *USBOLD*, *UNDERLINE*, or *ULBOLD*.

PLAIN The characters of the font are sent to the output device without modification.

UNDERSCORE With the exception of space characters, all characters of the font are underlined.

BOLD The characters of the font are bolded.

USBOLD With the exception of space characters, all characters of the font are underlined and bolded.

UNDERLINE The characters of the font are underlined.

ULBOLD The characters of the font are underlined and bolded.

15.10.5 FONTPAUSE Block

A change from one font to another font often requires WATCOM Script/GML to send a control sequence to the output device. In some cases, the font switch may require physical intervention at the output device by the operator. Examples of such an intervention would be changing a print wheel or color ribbon. The **fontpause** block defines a pausing method to use, and is referenced by the device font block. The font pause block is not required if font pausing is not needed.

```
:FONTPAUSE
   type  = 'character string'
   :value.
      <device functions>
   :evalue.
:eFONTPAUSE.
```

Figure 131. *The FONTPAUSE Block*

A fontpause block begins with the **:fontpause** tag and ends with the **:efontpause** tag. The type attribute and the value section must both be specified.

```
:FONTPAUSE
   type  = '12-pitch wheel'
   :value.
      %text( "Please attach the 12 pitch wheel" )
      %recordbreak()%wait()
   :evalue.
:eFONTPAUSE.
```

Figure 132. *Example of the FONTPAUSE Block*

15.10.5.1 TYPE Attribute

The **type** attribute specifies a character value. The specified value is referenced by the device font block when a font pause method is required. The attribute value must be unique among the font pause blocks in the device definition.

15.10.5.2 VALUE Section

The **value** section specifies the pausing control sequences to be output before the font switch, and is started with the **:value** tag. Device functions are then entered after the :value tag, and may be specified on more than one line. The **:evalue** tag delimits the end of a value section, and must be the first non-space characters in the line.

15.10.6 RULE Block

The **rule** block specifies the information necessary for WATCOM Script/GML to create a rule line, and must be specified.

```
:RULE
   font         = number | 'character string'
   rule_value   = number | 'character'
:eRULE.
```
Figure 133. *The RULE Block*

A rule block begins with the **:rule** tag and ends with the **:erule** tag. Both of the attribute values in the rule block must be specified.

```
:RULE
   font         = 0
   rule_value   = '-'
:eRULE.
```
Figure 134. *Example of the RULE Block*

15.10.6.1 FONT Attribute

The **font** attribute value may be either a non-negative integer number or a character value. If a number is specified, the font of the rule line will be the default font with the corresponding font number. A character attribute value must be a font name defined in a device font block. The font style PLAIN will be used in this case. Using a specific font name will ensure that the rule line will be drawn with a particular font regardless of the selected default fonts.

15.10.6.2 RULE_VALUE Attribute

The **rule_value** attribute may be either a number or a character value. The number value may be either a non-negative integer or a hexadecimal number beginning with a dollar($) sign. The number represents a character available in the font being used for the rule line, and is usually used when the character cannot be entered at the terminal. If a character value is specified, it must be delimited by quotes and only one character in length. The attribute value is used to construct the rule line.

15.10.7 BOX Block

The **box** block specifies the information necessary for WATCOM Script/GML to create a box, and must be specified.

```
:BOX
   font              = number | 'character string'
   top_line          = number | 'character'
   bottom_line       = number | 'character'
   left_side         = number | 'character'
   right_side        = number | 'character'
   top_left          = number | 'character'
   top_right         = number | 'character'
   bottom_left       = number | 'character'
   bottom_right      = number | 'character'
:eBOX.
```

Figure 135. *The BOX Block*

A box block begins with the **:box** tag and ends with the **:ebox** tag. All of the box block attributes must be specified. With the exception of the font attribute, all of the attribute values may be either a number or a character value. The number value may be either a non-negative integer or a hexadecimal number starting with a dollar($) sign. The number represents a character available in the font being used for the box, and is usually used when the value cannot be entered as a character at the terminal. If a character value is specified, it must be delimited by quotes and only one character in length.

```
:BOX
   font         = 0
   top_line     = '-'
   bottom_line  = '-'
   left_side    = '!'
   right_side   = '!'
   top_left     = '+'
   top_right    = '+'
   bottom_left  = '+'
   bottom_right = '+'
:eBOX.
```

Figure 136. *Example of the BOX Block*

15.10.7.1 FONT Attribute

The **font** attribute may be either a non-negative integer number or a character value. If a number is specified, the font of the box will be the default font with the corresponding font number. A character attribute value must be a font name defined in a device font block. Using a specific font name will ensure that the box will be drawn with a particular font regardless of the selected default fonts.

15.10.7.2 TOP_LINE Attribute

The **top_line** attribute value specifies the character used to create the top line of the box.

15.10.7.3 BOTTOM_LINE Attribute

The **bottom_line** attribute value specifies the character used to create the bottom line of the box.

15.10.7.4 LEFT_SIDE Attribute

The **left_side** attribute value specifies the character used to create the left side line of the box.

15.10.7.5 RIGHT_SIDE Attribute

The **right_side** attribute value specifies the character used to create the right side line of the box.

15.10.7.6 TOP_LEFT Attribute

The **top_left** attribute value specifies the character used to create the top left corner of the box.

15.10.7.7 TOP_RIGHT Attribute

The **top_right** attribute value specifies the character used to create the top right corner of the box.

15.10.7.8 BOTTOM_LEFT Attribute

The **bottom_left** attribute value specifies the character used to create the bottom left corner of the box.

15.10.7.9 BOTTOM_RIGHT Attribute

The **bottom_right** attribute value specifies the character used to create the bottom right corner of the box.

15.10.8 UNDERSCORE Block

The **underscore** block specifies the character information needed by WATCOM Script/GML to perform underscoring. If the underscore block is not specified, the standard underscore character will be used.

```
:UNDERSCORE
   font         = number | 'character string'
   score_value  = number | 'character'
:eUNDERSCORE.
```

Figure 137. *The UNDERSCORE Block*

An underscore block begins with the **:underscore** tag and ends with the **:eunderscore** tag. All of the attributes in the underscore block must be specified.

```
:UNDERSCORE
   font         = 0
   score_value  = '_'
:eUNDERSCORE.
```

Figure 138. *Example of the UNDERSCORE Block*

15.10.8.1 FONT Attribute

The **font** attribute value may be either a non-negative integer number or a character value. If a number is specified, the font of the underscore character will the default font with the corresponding font number. A character attribute value must be a font name defined in a device font block. The font style PLAIN will be used in this case. Using a specific font name will ensure that the underscore character will be used with a particular font regardless of the selected default fonts.

15.10.8.2 SCORE_VALUE Attribute

The **score_value** attribute may be either a number or a character value. The number value may be either a non-negative integer or a hexadecimal number beginning with a dollar($) sign. The number represents a character available in the font being used for the underscore character, and is usually used when the character cannot be entered at the terminal. If a character value is specified, it must be delimited by quotes and only one character in length. The specified attribute is used to underscore text.

15.10.9 PAGESTART Block

The **pagestart** block specifies the address of the first line on the output page.

```
:PAGESTART
   x_start  = number
   y_start  = number
:ePAGESTART.
```

Figure 139. *The PAGESTART Block*

A pagestart block begins with the **:pagestart** tag and ends with the **:epagestart** tag. All of the attributes in the pagestart block must be specified.

```
:PAGESTART
   x_start  = 300
   y_start  = 500
:ePAGESTART.
```

Figure 140. *Example of the PAGESTART Block*

15.10.9.1 X_START Attribute

The **x_start** attribute value is a non-negative integer number. The value is in horizontal base units.

15.10.9.2 Y_START Attribute

The **y_start** attribute value is a non-negative integer number. The value is in vertical base units.

16 Running WATCOM GENDEV

This section describes how you invoke WATCOM GENDEV and the options that may be specified.

WATCOM GENDEV is invoked by entering:

```
GENDEV file-name options
```

The "file-name" specifies the file containing the device, font and/or driver definitions. If the file type part of the file name (see "Files" on page 281) is not specified, WATCOM GENDEV searches for source files with the default file type for device and driver definitions. The font definition file type is the default alternate extension.

File Type	***Definition (IBM VM/CMS)***
VMD	default file type for the device and driver definition.
FONT	default file type for the font definition.
COPY	default file type for the created member name.

File Type	***Definition (IBM PC/DOS)***
PCD	default file type for the device and driver definition.
FON	default file type for the font definition.
COP	default file type for the created member name.

File Type	***Definition (DEC VAX/VMS)***
VXD	default file type for the device and driver definition.
FON	default file type for the font definition.
TXT	default file type for the created member name.

16.1 Options

The following options may be specified on the WATCOM GENDEV command line. The options are illustrated with an example showing the format for each of the computer systems supported by WATCOM GENDEV. With each option, upper case letters are used to indicate the minimum number of characters that must be specified.

Refer to "Running WATCOM Script/GML" on page 207 for more information about specifying command lines on your system.

16.1.1 ALTEXTension

```
IBM VM/CMS and IBM PC/DOS
    ALTEXT dev
DEC VAX/VMS
    /ALTEXT=dev
```

When a GENDEV source file is specified on the GENDEV command line, or as an include file, the file type may be omitted. A default file type will be supplied by WATCOM GENDEV. If the source file cannot be found with the default file type, the alternate extension option supplies a second file type to find with the source file.

16.1.2 DELim

```
IBM VM/CMS and IBM PC/DOS
    DEL #
DEC VAX/VMS
    /DEL=#
```

The value of the delimiter option is a single character. The delimiter value is used in the definition as the tag delimiter in place of the colon character.

16.1.3 INCList/NOINCList

```
IBM VM/CMS and IBM PC/DOS
    INCL
    NOINCL
DEC VAX/VMS
    /INCL
    /NOINCL
```

The INCLIST option displays on the terminal the name of each source file as it is included. The name of each include file is not displayed on the terminal as it is included when the NOINCLIST option (the default) is specified.

16.1.4 WARNing/NOWARNing

```
IBM VM/CMS and IBM PC/DOS
    WARN
    NOWARN
DEC VAX/VMS
    /WARN
    /NOWARN
```

The WARNING option (the default) causes WATCOM GENDEV warning messages about possible error situations to be displayed on the screen. Processing of the definition is not halted when a warning message is displayed. WATCOM GENDEV warnings about possible error situations are not displayed on the screen when the NOWARNING option is specified.

17 Files

17.1 Introduction

This chapter introduces the concept of *files* and *output devices* which are used to store and display data. For example, a disk can be used to store a file containing document text. A device such as a printer can also be treated as if it were a file, although it is only useful for displaying data; an attempt to read information from this device is invalid.

A software system such as the WATCOM Editor or WATCOM Script/GML can access a number of devices. Some devices (such as disks) can be used to store a number of files. Other devices (such as printers or a screen) have limited, special purpose uses.

17.2 File Specification

The general format of a file specification is as follows (items enclosed in brackets([]) are optional):

```
[(attribute[:attribute...])]file-designation
```

There are two components of a file specification. The attribute component describes the records which are in the file. The file-designation component describes the location and name of the file.

17.3 Files with IBM PC/DOS

This section describes the specification of a file on an IBM PC/DOS system. The attributes of a file are not stored with the file on the IBM PC/DOS system. A file created with a particular set of attributes must therefore have those attributes specified when that file is later referenced. It should also be noted that there is an upper limit on the number of open files. The default system limit can be increased with the FILES command in the CONFIG.SYS file. See your DOS manual for more information.

17.3.1 Record Attributes

The attributes are specified inside a pair of parentheses and must precede the file designation with no intervening spaces. The attributes are all optional. When specified, attributes must be separated by a colon. Non-numeric attributes can be abbreviated by truncating characters from the end to a minimum of one character.

17.3.1.1 Record Type

A file should be viewed as a number of *records*, where each record is a sequence of zero or more characters. WATCOM Script/GML supports files with three different types of records.

Text A *text* file consists of variable length records. Some of the possible character values cannot normally be entered at the keyboard. Text files are most commonly used for containing document source and other human-readable data. Two special characters are used to signify the end of a record. The CR (carriage return) and LF (line feed) characters separate records in a text file. These characters are automatically added to the end of each record, and should not be accounted for when determining the appropriate record size for the file. The record size of a text file specifies the size of the largest record which may be read from or written to that file.

Variable A *variable* file consists of variable length records. A variable file may contain any possible character. A 16 bit number at the beginning of the record specifies the length of each record. This number is automatically added to the beginning of each record, and should not be accounted for when determining the appropriate record size for the file. The record size of a variable file specifies the size of the largest record which may be read from or written to that file.

Fixed A *fixed* file consists of fixed length records. A fixed file may contain any possible character. The record size of a fixed file specifies the size of each record read from or written to that file.

The default record type of a document source file read by WATCOM Script/GML is *text*. The default record type of the WATCOM Script/GML output file or device is determined by the device definition being used.

17.3.1.2 Record Size

The record size attribute is a sequence of numeric digits which specifies the record length for the file. A record which is longer than the specified record size will be truncated. A record size of 132 is the default record size of a document source file read by WATCOM Script/GML. The default record size of the WATCOM Script/GML output file or device is determined by the device definition being used.

17.3.2 File Designation

A file designation may be any valid filename recognized by the IBM PC/DOS system. In general, a file designation looks like:

```
drive:\path\filename.ext
```

drive: If the *drive name* is omitted, the default drive is assumed. Examples of drive names are:

A:	disk drive A
B:	disk drive B
C:	disk drive C

\path If the *path* specification is omitted, the current directory is used. The path may be used to refer to files that are stored in sub-directories of the disk. Some examples of path specifications are:

```
\top\
\gml\data\
..\tests\
```

Your IBM PC/DOS manual can tell you more about directories: how to create them; how to store files in them; how to specify a path; etc.

filename The filename may contain up to eight characters, and is the main part of the file's name. If more than eight characters are used, only the first eight are meaningful. This is an important point. IBM PC/DOS does not check that the name is too long. If you specify more than eight characters then you may inadvertently destroy an existing file whose name happens to match the first eight characters.

.ext The file *extension* is an optional one to three character value which is a convenience in classifying files. The extension may only be used with disk file names. If it is specified, the period character separates the extension from the filename. The experienced user

will specify the file extension to identify the type of information stored in the file. The files of source text for WATCOM Script/GML usually have **gml** as the file extension.

17.3.3 Special Device Names

IBM PC/DOS has reserved some names for devices. These special device names are:

CON	the console (or terminal)
AUX	the serial port
COM1	serial port 1
COM2	serial port 2
PRN	the parallel printer
LPT1	the first parallel printer
LPT2	the second parallel printer
LPT3	the third parallel printer
NUL	nonexistent device

When using one of these special device names, no other part of the file specification may be specified. Earlier versions of DOS allowed a trailing ":" to be specified after a special device name. Starting with DOS 2.0, a trailing ":" may not be specified. For example, "CON" is acceptable but "CON:" is not.

17.3.4 File Specification Examples

The following are some examples of a valid file specification.

1. The following file specification refers to a file in the current directory of the default disk.

```
DATA.FIL
```

2. The file specification below indicates that the file is to have fixed-length records of length 130.

```
(F:130)EXAMPLE1.TST
```

3. The file specification below indicates that the file is to have variable-length records of maximum length 145, and resides on the "C" disk.

```
(V:145)C:NOVEMBER.RPT
```

4. The file specification below indicates that the file resides in the "RECORDS" directory of the "B" disk.

```
b:\records\bigmanual.gml
```

Note that the trailing "l" in the file name will be ignored. Thus the following designation is equivalent.

```
b:\records\bigmanua.gml
```

5. The file specification below refers to a second parallel printer.

```
LPT2
```

17.4 Files with IBM VM/CMS

This section describes the specification of a file on the IBM VM/CMS system.

17.4.1 Record Attributes

The attributes are specified inside a pair of parentheses and must precede the file designation with no intervening spaces. The attributes are all optional. When specified, attributes must be separated by a colon. Non-numeric attributes can be abbreviated by truncating characters from the end to a minimum of one character.

17.4.1.1 Record Type

A file should be viewed as a number of *records*, where each record is a sequence of one or more characters. WATCOM Script/GML supports files with three different types of records.

Text A *text* file consists of variable length records. Some of the possible character values cannot normally be entered at the keyboard. Text files are most commonly used for containing document source and other human-readable data. The record size of a text file specifies the size of the largest record which may be read from or written to that file.

Variable A *variable* file consists of variable length records. A variable file may contain any possible character. The record size of a variable file specifies the size of the largest record which may be read from or written to that file.

Fixed A *fixed* file consists of fixed length records. A fixed file may contain any possible character. The record size of a fixed file specifies the size of each record read from or written to that file.

The default record type of a document source file read by WATCOM Script/GML is *text*. The default record type of the WATCOM Script/GML output file or device is determined by the device definition being used.

17.4.1.2 Record Size

The record size attribute is a sequence of numeric digits which specifies the record length for the file. A record which is longer than the specified record size will be truncated. A record size of 132 is the default record size of a document source file read by WATCOM Script/GML. The default record size of the WATCOM Script/GML output file or device is determined by the device definition being used.

17.4.2 File Designation

A file designation may be any valid filename recognized by the IBM VM/CMS system. In general, a file designation looks like:

```
filename filetype filemode
```

filename The *filename* may contain up to eight characters, and is the main part of the file's name. If more than eight characters are used, only the first eight are meaningful. This is an important point. IBM VM/CMS does not check that the name is too long. If you specify more than eight characters then you may inadvertently destroy an existing file whose name happens to match the first eight characters.

filetype The *filetype* is a one to eight character value which is a convenience in classifying files. The file type may only be used with disk file names. If it is specified, the space character separates the file type from the filename. The file type may be omitted from a disk file name, but the experienced user will probably always use it. The default file type is **file** if it is not specified. The files of source text for WATCOM Script/GML usually have **gml** as the file type.

filemode The *filemode* is an optional two character value which specifies the CMS mini-disk containing the file. The first character is a letter identifying the mini-disk. The second character is a mode number for the file. The file mode may only be used with disk file names. If it is specified, the space character separates the file mode from the filetype.

If the file mode is not specified, then it will default to "*". For a new file, this will result in a file mode of "A1". For an existing file, "*" will match the first mini-disk that contains a file with the

specified file name and type. You should consult the appropriate CMS documentation for more information on the file mode.

The components of a file designation are case insensitive. Upper and lower case letters are treated identically. For example, the following file designations are equivalent and refer to the same file.

```
Book Gml
BOOK GML
book gml
```

Each component of the file designation is separated from the next by one or more space characters. WATCOM Script/GML also allows a period to separate the components of the file designation. The following file designations refer to the same file.

```
book gml a
book.gml a
book.gml.a
```

17.4.3 Special File Names

Some file names have been reserved for devices. These special names are:

TERMINAL	the terminal screen
TERM	the terminal screen
READER	the VM virtual reader
RDR	the VM virtual reader
PRINTER	the VM virtual printer
PRT	the VM virtual printer
PUNCH	the VM virtual punch
PUN	the VM virtual punch

When using one of these special file names, no other part of the file specification may be specified.

17.4.4 File Specification Examples

The following are some examples of a valid file specification.

1. The following file specification refers to a file on the current mini-disk.

   ```
   DATA.FIL
   ```

2. The file specification below indicates that the file is to have fixed-length records of length 130.

```
(F:130)EXAMPLE1.TST
```

3. The file specification below indicates that the file is to have variable-length records of maximum length 145, and resides on the "C" mini-disk.

```
(V:145)NOVEMBER.RPT.C
```

4. The file specification below indicates that the file resides on the ""C" mini-disk.

```
testables.dat.c
```

Note that the trailing "s" in the file name will be ignored. Thus the following designation is equivalent.

```
testable.dat.c
```

5. The file specification below refers to the terminal device.

```
TERMINAL
```

17.5 Files with DEC VAX/VMS

This section describes the specification of a file on the DEC VAX/VMS system.

17.5.1 Record Attributes

The attributes are specified inside a pair of parentheses and must precede the file designation with no intervening spaces. The attributes are all optional. When specified, attributes must be separated by a colon. Non-numeric attributes can be abbreviated by truncating characters from the end to a minimum of one character.

17.5.1.1 File Type

Carriage The *carriage* attribute specifies the file as having ASA (American Standards Association) carriage control characters. A control character begins each record and is used for vertical spacing control. The VAX VMS system saves the carriage control attribute with the file. The attribute is later used when printing the file to indicate to the system that the carriage control characters should be used for vertical spacing. The valid characters and their interpretation are:

'1'	Advance to the top of the next page.
'+'	Advance zero lines (overprint).
' '	Advance one line.

'0'	Advance two lines.
'-'	Advance three lines. This carriage control character is not supported by some printers. It should only be used when the printer is known to support ASA triple spacing.

17.5.1.2 Record Type

A file should be viewed as a number of *records*, where each record is a sequence of zero or more characters. WATCOM Script/GML supports files with three different types of records.

Text	A *text* file consists of variable length records. Some of the possible character values cannot normally be entered at the keyboard. Text files are most commonly used for containing document source and other human-readable data. The record size of a text file specifies the size of the largest record which may be read from or written to that file.
Variable	A *variable* file consists of variable length records. A variable file may contain any possible character. The record size of a variable file specifies the size of the largest record which may be read from or written to that file.
Fixed	A *fixed* file consists of fixed length records. A fixed file may contain any possible character. The record size of a fixed file specifies the size of each record read from or written to that file.

The default record type of a document source file read by WATCOM Script/GML is *text*. The default record type of the WATCOM Script/GML output file or device is determined by the device definition being used.

17.5.1.3 Record Size

The record size attribute is a sequence of numeric digits which specifies the record length for the file. A record which is longer than the specified record size will be truncated. A record size of 132 is the default record size of a document source file read by WATCOM Script/GML. The default record size of the WATCOM Script/GML output file or device is determined by the device definition being used.

17.5.2 File Designation

A file designation may be any valid filename recognized by the VMS Record Management Services (RMS). In general, a file designation looks like:

```
node::device:[directory]filename.type;version
```

node:: The *node::* is the DECnet node name (This feature may not be installed on your VAX/VMS system).

device: The *device:* is the *device* name. The device name is optional and defaults to the disk containing your current directory. The device name is followed by a ":". Usually, the device name is not present because most files are located on the default disk. Some of the device names on the VAX/VMS system are:

DRA1:	A disk drive.
MSA0:	A magtape drive.
TT:	The screen/keyboard.
LPA0:	The printer.

Devices may have logical names. It is usually more convenient to refer to a disk by its logical name. Some examples of logical names are:

SYS$SYSDEVICE:
SYS$SYSROOT:
DISK$CUSTOMER:

The logical name can be 1 to 63 characters (letters A-Z, $, and/or numbers 0-9 only) in length.

[directory] The file is located in the specified "directory". The default is the current directory; consequently it is often not specified. Directory names have up to nine characters. If a file is contained in a subdirectory of another directory, the subdirectory name is specified following the directory name. These names are separated from each other by a period. A minus sign at the start of a directory specification indicates the parent directory of the current directory. The directory name, including any subdirectories, is enclosed by square brackets ([]). Some examples of directory names are:

```
[USERFILES]
[EXAMPLE.PROGRAMS]
[-.reports]
```

filename The *filename* is the main part of the file's name. File names have one to thirty-nine characters (letters A-Z and/or numbers 0-9 only). Some examples of file names are:

```
TEST1
setup
MSGBOARD
```

.type The *type* is an optional file type consisting of one to thirty-nine characters (letters A-Z and/or numbers 0-9 only). The extension may only be used with disk file names. If it is specified, the period(.) character separates the extension from the filename. The file extension may be omitted from a disk file name, but the experienced user will probably always use it. The files of source text for WATCOM Script/GML usually have **gml** as the file extension.

;version The *version* is the version number of the file. Any time you create a new file that has the same name as a file already in the directory, the system automatically selects a version number that is one greater than the last version number. If you have more than one file with the same name, you can explicitly identify which file you want by its version number. The latest version of a file is used if the version number is not specified. The version number is preceded by a semicolon(;). Some examples of file designations with their version numbers are:

```
SAMPLE.DAT;1
SAMPLE.DAT;2
REPORT.GML;15
TRIAL.;4
[LIBRARY]WEDIT.EXE;1
```

Most of the fields are optional. For example, a file under the current default directory can be referenced without specifying a node name, device name, or directory, and often without a version number. Devices such as a printer can be referenced without specifying any fields other than the device name.

In general, the different components of a file designation are made up of the letters A through Z and the digits 0 through 9. Component specifications are case insensitive. Upper and lower case letters are treated identically. For example, the file designations

```
Sample.Dat
SAMPLE.DAT
sample.dat
```

are equivalent.

17.5.3 Writing to the Printer

Output can be written to a printer by using the printer device name as a file name. A better way of doing this (from the point of view of document portability) is to define the name "printer" using the VMS DEFINE command prior to invoking the software, and then to use "printer" as a device name. The device name "printer" is used by the software on most computer systems to designate the system printer. The required DEFINE could appear in a "LOGIN" command file, and would be similar to:

```
DEFINE PRINTER LPA0
```

17.5.4 Using the Terminal as a File

The terminal can be used for input or output by opening a file whose name is the device name of the terminal. One alternative is to use the logical names SYS$INPUT and SYS$OUTPUT to refer to the terminal. Another possibility is to define a logical name "terminal" using the VMS DEFINE command prior to invoking the software. Use of the device name "terminal" provides a degree of portability, since this name is used by the software on most computer systems to designate the terminal. The required DEFINE command could appear in a "LOGIN" command file, and would be similar to:

```
DEFINE TERMINAL TT
```

17.5.5 File Specification Examples

The following are some examples of a valid file specification.

1. The following file specification refers to the file "data.gml" in the current default directory:

   ```
   data.gml
   ```

2. If the current default directory is [USERFILES], the following file specification will refer to the file FILE1.REP in the [EXAMPLES] directory:

   ```
   [examples]file1.rep
   ```

3. The file designation below indicates that the file resides on a logical device called "source" and in the directory called "document.text". Version number 5 of the file is specified:

   ```
   source:[document.text]book.gml;5
   ```

18 Libraries

When WATCOM Script/GML processes a document, it must have information about the output device and character sets being used. The WATCOM GENDEV program is used to create and modify this information (see "Creating a Definition" on page 228). The definition of each device and character set is saved in a separate file. These definition files are grouped together in a **library**. Each of the individual files is called a **member** of the library. Libraries are used by WATCOM Script/GML to localize the placement of definitions needed to process a document.

Since a member in a library is created by the WATCOM GENDEV program as a file, the length and character composition of the member name is limited by the file name restrictions of the computer system. To minimize this restriction, every definition has two names associated with it. The **member name** is the name of the file or library member which contains the definition. The **defined name** is the name used by WATCOM Script/GML and WATCOM GENDEV when referring to the definition. A defined name can be up to 78 characters in length.

When a defined name is referenced, the member name associated with that defined name must be known. This is accomplished through the use of a "directory" file which contains the defined name and its associated member name for each definition contained in the library. This file is named **WGMLST**, and is automatically created when the WATCOM GENDEV program is used to process a definition. The name WGMLST must not be used as a member name for any of the definitions.

More than one definition library may be defined. When there is a central place where a number of people share data, a library containing all common definitions may be shared. If an individual wishes to modify an existing definition without affecting the shared library, they may create a personal library containing their modifications. When a definition is required, the personal library will be searched first. If the definition is not found, then the shared library is searched. The number of search levels can be extended to the requirements of the people using the system.

Before WATCOM Script/GML or WATCOM GENDEV is invoked, a list of the library names must be defined. The search order for the libraries is from the first name in the list to the last name. When the WATCOM GENDEV program is used, this library list should only contain the name of the library you wish to create or update. In addition, WATCOM GENDEV will always try to find the WGMLST member file on the current directory or disk before trying to find it in a library. The method for specifying the

library list depends on the machine being used, and is discussed further in the following subsections.

18.1 Libraries with IBM VM/CMS

18.1.1 Creating and Updating a Library

A definition library is maintained with the use of the **MACLIB** command. A maclib is a file which contains the contents of many individual files. The definition files produced by the WATCOM GENDEV program are added to and deleted from a maclib. The file type of a maclib member file must be **COPY**. This is the default file type produced by WATCOM GENDEV. All maclib operations are in terms of the member name, not the defined name of a definition.

The **MACLIB GEN** command is used to create a maclib.

```
MACLIB GEN gmllib wgmlst
```
Figure 141. *Creating an IBM VM/CMS Library*

The first name after the command is the name of the library to be created. If this library already exists, it is erased first. The file type of the library will always be **MACLIB**. The second name after the command is the name of the member file to place in the library. Once a member is in a library, the member file may be erased.

The **MACLIB DEL** command is used to delete a member from a library.

```
MACLIB DEL gmllib qume
```
Figure 142. *Deleting an IBM VM/CMS Library Member*

The member in the *gmllib* maclib with the name *qume* is deleted.

The **MACLIB ADD** command is used to add a member to an existing library.

```
MACLIB ADD gmllib qume
```
Figure 143. *Adding an IBM VM/CMS Library Member*

The member file with the name *qume* is added to the *gmllib* maclib. If a member already exists in the maclib with the same name, it is NOT deleted from the maclib first.

There would then be two versions of the same member in the maclib. In this case, WATCOM Script/GML would find the old version. It is therefore important to ensure a member is first deleted from the maclib before a new version is added.

The maclib command is more completely described by the documentation available with the system.

18.1.2 Defining a Library List

The name of a definition library can be any arbitrary file name. To locate the library, WATCOM Script/GML and WATCOM GENDEV must have a list of library names. This list is defined with the **GLOBAL MACLIB** command.

```
GLOBAL MACLIB gmllib
```
Figure 144. *Defining the IBM VM/CMS Library List*

The global maclib command must be performed before either WATCOM Script/GML or WATCOM GENDEV are invoked. If *gmllib* is the shared library and *mylib* is the personal library, the following command specifies the proper library list.

```
GLOBAL MACLIB mylib gmllib
```
Figure 145.

18.2 Libraries with DEC VAX/VMS

18.2.1 Creating and Updating a Library

A definition library is maintained with the use of the VMS **LIBRARY** command. A library is a file which contains the contents of many individual files. The definition files produced by the WATCOM GENDEV program are added to and deleted from the library. The library member file can have an arbitrary file type. The default file type produced by WATCOM GENDEV is **TXT**. All library operations are in terms of the member name, not the defined name of a definition.

The **LIBRARY/TEXT/CREATE** command is used to create a library.

```
LIBRARY/TEXT/CREATE gmllib.tlb wgmlst.txt
```
Figure 146. *Creating a DEC VAX/VMS Library*

The first name after the command is the name of the library to be created. If this library already exists, a new version is created. The second name after the command is the name of the member file to place in the library. Once a member is in a library, the member file can be erased.

The **LIBRARY/TEXT/DELETE** command is used to delete a member from a library.

```
LIBRARY/TEXT/DELETE=qume gmllib.tlb
```
Figure 147. *Deleting a DEC VAX/VMS Library Member*

The member in the *gmllib* library with the name *qume* is deleted. Once a member is in the library, the file type is no longer needed.

The **LIBRARY/TEXT/INSERT** command is used to add a member to an existing library.

```
LIBRARY/TEXT/INSERT gmllib.tlb qume.txt
```
Figure 148. *Adding a DEC VAX/VMS Library Member*

The member file with the name *qume* is added to the *gmllib* library. If a member already exists in the library with the same name, the library command generates an error message. A member must first be deleted from the library before a new version is added.

The library command is more completely described by the documentation available with the system.

18.2.2 Defining a Library List

The name of a definition library can be any arbitrary file name. To locate the library, WATCOM Script/GML and WATCOM GENDEV must have a list of library names. This list is defined with the **ASSIGN** command.

```
ASSIGN <gml>gmllib.tlb GMLLIB:
```
Figure 149. *Defining the DEC VAX/VMS Library List*

The assign command must be performed before either WATCOM Script/GML or WATCOM GENDEV are invoked. The assigned name **GMLLIB:** must be used. If *gmllib* is the shared library and *mylib* is the personal library, the following command specifies the proper library list.

```
ASSIGN <dave>mylib.tlb,<gml>gmllib.tlb GMLLIB:
```
Figure 150.

18.3 Libraries with IBM PC/DOS

18.3.1 Creating and Updating a Library

A definition library is a specific directory on a device. Creating a directory on a disk and placing the member files in it is accomplished with standard IBM PC/DOS file commands.

18.3.2 Defining a Library List

To locate the library, WATCOM Script/GML and WATCOM GENDEV must have a list of library directories. This list is defined with the DOS **SET** command.

```
SET GMLLIB=A:\wgmlib\
```
Figure 151. *Defining the IBM PC/DOS Library List*

The set command must be performed before either WATCOM Script/GML or WATCOM GENDEV are invoked. The name **GMLLIB** must be used, and there should be no blanks between it and the equals sign. If *wgmlib* is the shared library directory and *mylib* is the personal library directory, the following command specifies the proper library list.

```
SET GMLLIB=A:\mylib\;A:\wgmlib\
```
Figure 152.

Each library directory is separated from the next by a semi-colon.

Appendices

APPENDIX A UnProcessed Script Control Words

bf	bs	df	du	eq
gg	hw	hy	hw	hy
it	oo	pf	ph	rc
sv	uw	zc		

APPENDIX B WATCOM Script/GML Error Messages

Some error messages are 'continued' by other messages. The ZZ error class contains the continuation messages, with references to them noted after the text of the starting error message.

A number of special symbols are used to insert information known only when the error occurs. The are:

%s	a text string replaces the %s
%d	a number replaces the %n
%n	a new line is started on the screen
%t	a tab to the screen position indicated by the number following the %t (zero(0) means the number is supplied when the error occurs)

The error messages are:

AT	***Attribute Messages***
AT--001	Required attribute not found
AT--002	Missing attribute value
AT--003	Invalid attribute value
AT--004	Attribute already found
AT--005	Missing or invalid closing quote on attribute
AT--006	Attribute invalid for current format
AT--007	The attribute value cannot have more than one character
AT--008	Missing required equals sign
AT--009	Illegal banner reference, banner does not exist
AT--010	Illegal banner region reference
AT--011	Cannot reference banner currently being defined
AT--012	If one of refdoc or refplace is specified both are required
AT--013	All attributes required when defining new banner
AT--014	All attributes required when defining new banner region
AT--015	Id attribute only valid for ordered list items
AT--016	Number value cannot be greater than 32767
AT--017	Number value must be greater than zero
AT--018	Horizontal space unit is too large
AT--019	Vertical space unit is too large
AT--020	The BOX and 'string' values are not supported with the %n frame attribute of the FN layout tag -- Setting to RULE

AT--021	The script_format attribute has been set to NO %n for the banner region with a keyword content value
AT--022	The ix attribute has a range of 1 through 9
AT--023	If a new content is specified for an existing banner %n region, the previous script_format value of yes %n must be re-specified

CL ***Command Line Messages***

CL--001	For '%s'%n Value is an invalid option in the command line
CL--002	Must have a device specification
CL--003	For format '%s'%n Missing or invalid format
CL--004	Missing option value for '%s'
CL--005	Name of command file is not specified
CL--006	The font number value must be between 1 and 255 inclusive
CL--007	Document source file specified more than once
CL--008	For passes option, value must be between 1 and 255 inclusive
CL--009	The FROM option value must be greater than zero
CL--010	The TO option value must be greater than zero
CL--011	For the FONT option value '%s'%n Number is too large or contains invalid characters
CL--012	The option file '%s' is recursively included
CL--013	The CPInch option value may not be less than zero

FN ***Footnote Messages***

FN--001	DELETED
FN--002	Footnote too large, it exceeded a page

LO ***Layout Messages***

LO--001	For banner with docsect = %s and place = %s *followed by the error message ZZ--01*
LO--002	For banner with docsect = %s and place = %s *followed by the error message ZZ--02*
LO--003	For banner with docsect = %s and place = %s *followed by the error message ZZ--03*
LO--004	For banner with docsect = %s and place = %s *followed by the error messages ZZ--04 and ZZ--08*
LO--005	For banner with docsect = %s and place = %s *followed by the error messages ZZ--04 and ZZ--07*
LO--006	For banner with docsect = %s and place = %s *followed by the error messages ZZ--04 and ZZ--05*
LO--007	For :H%d layout tag - a heading that forces%n a page eject must also force a line break
LO--008	For banner with docsect = %s and place = %s

followed by the error messages ZZ--04 and ZZ--06

LO--009 First page of letter cannot be even

LO--010 Left margin plus binding must be greater than zero

LO--011 For the number_form attribute (layout tag H%d) %n Should not have value 'prop' without having %n value 'new' for a higher heading level

LO--012 For :H%d layout tag, the indent attribute is%n greater than or equal to the line width

LO--013 For the level attribute (layout tag %s) %n Level number %d must be specified

LO--014 The layout right margin is greater %n than the device page width

LO--015 The right margin must be greater than the left margin

LO--016 The right margin is too small when adjusted %n for the uprintable area of the device page

LO--017 The page depth is too small when adjusted %n for the uprintable area of the device page

IN ***Informative messages***

IN--001 Processing layout

IN--002 Formatting document

IN--003 Formatting complete

IN--004 Number of tags processed: %t40%s

IN--005 Number of words processed: %t40%s

IN--006 Number of source lines processed: %t40%s

IN--007 Number of files included: %t40%s

IN--008 Number of unformatted lines processed: %t40%s

IN--009 Number of headings processed: %t40%s

IN--010 Total size of headings processed: %t40%s

IN--011 Number of footnotes processed: %t40%s

IN--012 Number of index entries created: %t40%s

IN--013 Number of text lines produced: %t40%s

IN--014 Number of pages produced: %t40%s

IN--015 Time to process: %t40%s.%d

IN--016 DELETED

IN--017 pass #%d

IN--018 %t0%s

IN--019 Current file is '%s'

IN--020 Number of output records produced: %t40%s

IN--021 Page %d specified by the TO option has been processed

IN--022 Processing device information

IO ***I/O Messages***

IO--001 For file '%s'

followed by the error messages ZZ--09 and ZZ--10

IO--002 Having more than %s hundred file includes and macro %n invocations is probably the result of recursion
IO--003 Invalid imbed file
IO--004 System message is '%s'%n Error number is %d%n Output operation failed
IO--005 System message is '%s'%n Error reading input file
IO--006 GML interrupted by ATTN key
IO--007 The device directory or library%n '%s'%n does not exist
IO--008 For the device (or font) '%s':%n The information file for this name cannot be found.%n If the device/font has been defined, the problem may%n be that the DOS SET symbol GMLLIB has not been%n correctly set to point to the device library.
IO--009 Error reading device/font library
IO--010 Error finding device/font member
IO--011 Output file's record size is too small for%n the device '%s'
IO--012 The valueset file has no value records
IO--013 Error reading device/font library%n Member name is '%s'
IO--014 Error opening virtual page file %n probable causes are a full disk or no more file handles
IO--015 .SY failed %n System message is '%s'

IX ***Index Messages***

IX--001 Id not defined
IX--002 Referencing not allowed for :I1 tag
IX--003 Parent index entry not defined
IX--004 Id already defined
IX--005 Major page number reference already defined
IX--006 Page range incomplete
IX--007 Page range starting entry not allowed inside floating block
IX--008 Page range starting entry already specified
IX--009 No page range starting entry specified
IX--010 Index option must be specified to process index tags

RF ***Referencing Messages***

RF--001 For id '%s'%n Figure id not defined
RF--002 For id '%s'%n More passes required for figure referencing
RF--003 For id '%s'%n Heading id not defined
RF--004 For id '%s'%n More passes required for heading referencing
RF--005 More passes required for TOC or FIGLIST
RF--006 For id '%s'%n More passes required for footnote referencing
RF--007 For id '%s' Duplicate figure id
RF--008 For id '%s' Duplicate heading id

RF--009	For id '%s' Duplicate footnote id
RF--010	For id '%s'%n Footnote id not defined
RF--011	Two passes needed to get TOC or FIGLIST %n in front material
RF--012	For id '%s'%n Duplicate list item id
RF--013	For id '%s'%n List item id not defined
RF--014	For id '%s'%n More passes required for list item referencing
RF--015	For id '%s'%n Identifier name should have no more than %n seven characters
RF--016	For the character '%c' in the id '%s'%n Identifier name should consist of letters and numbers

SN ***Syntax Messages***

SN--001	Number is too large or contains invalid characters
SN--002	Missing or invalid filename
SN--003	Expecting a layout tag or attribute, found text
SN--004	Expecting text, found a GML tag %n or Script control word/macro
SN--005	%s cannot be in a footnote, figure, example,%n floating block, floating keep, conditional column,%n or conditional page
SN--006	Symbol does not exist, cannot delete it
SN--007	Expecting EBANNER tag
SN--008	Expecting EBANREGION tag
SN--009	For a BINCLUDE %s%n the reposition attribute must equal START
SN--010	Tag not supported in this version
SN--011	Left and right margins are too close together
SN--012	Expecting a GDOC tag or the end of the file
SN--013	Expecting a FRONTM, BODY, APPENDIX or BACKM tag
SN--014	Expecting the end of the file
SN--015	Expecting ETITLEP tag
SN--016	Expecting EADDRESS tag
SN--017	Expecting DD tag
SN--018	Expecting EDL tag
SN--019	Expecting EXMP tag
SN--020	Expecting EFIG tag
SN--021	Expecting EFN tag
SN--022	Expecting ESL tag
SN--023	Expecting EOL tag
SN--024	Expecting EUL tag
SN--025	Expecting ELQ tag
SN--026	Expecting ECIT tag
SN--027	Expecting EHP1 tag
SN--028	Expecting EHP2 tag
SN--029	Expecting EHP3 tag
SN--030	Expecting EQ tag
SN--031	Invalid heading level

SN--032	No heading allowed
SN--033	Expecting EGDOC tag
SN--034	Expecting EHP0 tag
SN--035	No text in input file
SN--036	Missing DT tag to precede DD tag
SN--037	Missing DTHD tag to precede DDHD tag
SN--038	Expecting DDHD tag
SN--039	Date already defined
SN--040	Document number already defined
SN--041	ATTN tag already specified
SN--042	Missing GT tag to precede GD tag
SN--043	Expecting OPEN tag
SN--044	Expecting CLOSE tag
SN--045	Expecting EDISTRIB tag
SN--046	Expecting ECLOSE tag
SN--047	Nesting of PSC tags is illegal
SN--048	Expecting EPSC tag
SN--049	Heading level(s) omitted: %n Encountered :H%d, expecting :H%d
SN--050	CMT tag should appear at start of line
SN--051	GRAPHIC output device will not support grey %n scales -- black and white image will be produced
SN--052	Figure with an Id must have a caption
SN--053	Example is too wide for the column
SN--054	Figure is too wide
SN--055	Invalid header information%n in the GKS pixel graphic file
SN--056	For a multiple column figure, place attribute %n must equal TOP
SN--057	The symbol name must have at least one character
SN--058	PSC tag not found
SN--059	Expecting ELAYOUT tag
SN--060	Expecting EGL tag
SN--061	Expecting GD tag
SN--062	DELETED
SN--063	Output line is too large to fit on a page
SN--064	Device/font library was created with a %n different version of GENDEV
SN--065	For font switch ’%s’%n Font switch name is not defined
SN--066	For font ’%s’%n Font name is not defined as a device font
SN--067	Width for space character should not be zero
SN--068	DELETED
SN--069	Width for :box character should not be zero
SN--070	Width for :underscore character should not be zero
SN--071	Depth of BINCLUDE file is greater than a page
SN--072	BINCLUDE tag must appear after the GDOC tag
SN--073	For the symbol ’%s’%n The character ’%c’ is not valid

SN--074 For the symbol '%s'%n The length of a symbol name may not exceed %n ten characters
SN--075 :LI or :LP tag must follow the start of a list
SN--076 Heading is too large for the output page
SN--077 Expecting BANNER tag
SN--078 Expecting a GML tag, found text
SN--079 Cannot set more than 999 symbols with the 'valueset' option
SN--080 SUBJECT tag already specified
SN--081 Cannot fit the text of a list item in the%n adjusted left and right margins
SN--082 Cannot fit the figure in the%n adjusted left and right margins
SN--083 Cannot fit the figure with a frame in the%n adjusted left and right margins
SN--084 Expecting ESF tag
SN--085 The GRAPHIC depth is greater than the page depth
SN--086 GRAPHIC tag must appear after the GDOC tag
SN--087 The GRAPHIC depth must be greater than zero
SN--088 The '%s' device cannot scale a GRAPHIC
SN--089 For device '%s', the GRAPHIC scale must%n be a multiple of 50 -- rounding will occur
SN--090 For default font %d, the font_height attribute %n must be specified when the font is scaled.
SN--091 Depth attribute required after GRAPHIC tag
SN--092 The GRAPHIC width is greater than the page width
SN--093 The GRAPHIC width must be greater than zero
SN--094 The GRAPHIC y-offset value is beyond the edge%n of the image -- none of the image can be printed
SN--095 The GRAPHIC x-offset value is beyond the edge%n of the image -- none of the image can be printed
SN--096 For device '%s', the GRAPHIC scale is too%n small -- setting scale to %d
SN--097 For device '%s', the GRAPHIC scale is too%n large -- setting scale to %d
SN--098 For font style '%s'%n Font style name is not defined
SN--099 More than %s thousand symbol substitutions on%n one record is probably the result of recursion

SC ***Script processing errors***

SC--001 A control word parameter is required
SC--002 The control word parameter '%s' is invalid
SC--003 A macro is not being defined
SC--004 End of file reached %n macro/remote '%s' is still being defined
SC--005 Macro '%s' is not being defined
SC--006 Unrecognized control word

SC--007	Expecting %s END
SC--008	A tag name must be specified
SC--009	The tag name is too long
SC--010	The tag operation is missing
SC--011	The TEXTERR option conflicts with the %n TEXTDEF, TEXTLINE and TEXTREQD options
SC--012	'%s' is an invalid tag operand
SC--013	User tag '%s' already exists
SC--014	User tag '%s' has not been defined
SC--015	Macro/Remote name is missing or invalid
SC--016	A tag is not currently being defined
SC--017	An attribute name must be specified
SC--018	The attribute name is too long
SC--019	An attribute is not currently being defined
SC--020	'%s' is an invalid attribute operand
SC--021	All attribute options except OFF, ON, UPPERCASE %n and RESET conflict with the AUTOMATIC option
SC--022	A new .GA control word must be used to %n specify more attribute options after '%s'
SC--023	The LENGTH option conflicts with the ANY, AUTOMATIC, %n RANGE, VALUE and RESET options
SC--024	The RANGE option conflicts with the ANY, AUTOMATIC, %n LENGTH and VALUE options
SC--025	Missing numeric operand
SC--026	Value name is missing or too long
SC--027	Value '%s' has already been specified
SC--028	USE operation must be specified before DEFAULT
SC--029	'%s' is only valid after the VALUE option
SC--030	Only USE or DEFAULT is valid after the VALUE option
SC--031	RESET option only applies to an ANY, AUTOMATIC, %n RANGE or VALUE attribute
SC--032	VALUE '%s' has not been defined
SC--033	Maximum value is smaller than the minimum
SC--034	The specified default is outside the range values
SC--035	AUTOMATIC option must have a defined value
SC--036	The tag '%s' has been deleted by %n a previous GT control word
SC--037	The macro '%s' for the gml tag '%s' %n is not defined
SC--038	Tag text may not be specified for the '%s' tag
SC--039	Tag text must be specified with the '%s' tag
SC--040	'%s' is not a valid attribute name
SC--041	Cannot specify the automatic attribute '%s'
SC--042	A value must be specified for the attribute '%s'
SC--043	Value for '%s' exceeds maxium length of %d
SC--044	Value for '%s' must be in the range %d to %d
SC--045	Value '%s' for the '%s' attribute is not defined

SC--046	The TEXTREQD and TEXTDEF options conflict
SC--047	For the tag '%s', the required attribute(s) *followed by the error message ZZ--16*
SC--048	A control word parameter is missing
SC--049	A single character or a two character hexadecimal %n value must be specified
SC--050	.IF cannot be nested more than 10 levels
SC--051	.TH must follow a .IF
SC--052	.EL must follow a .IF, .TH or .DO END
SC--053	Parameter to .DO must be BEGIN or END
SC--054	.DO BEGIN must follow a .IF, .TH or .EL
SC--055	.DO must be specified within a .IF structure
SC--056	Expecting a .DO BEGIN
SC--057	Tab character defined by .TB SET must be one character
SC--058	Right string delimeter missing
SC--059	Invalid text before tab position
SC--060	Invalid text after tab information
SC--061	%s positions not in ascending order
SC--062	Invalid %s position
SC--063	Too many operands in .BX command
SC--064	%s must have one of operands BEGIN, END or DUMP
SC--065	Too many operands in %s BEGIN control word
SC--066	%s END not preceded by %s BEGIN
SC--067	'%s' is an invalid tag operation
SC--068	missing relational operator
SC--069	invalid relational operator
SC--070	require a second operand for the relational expression
SC--071	The tag '%s' was not defined with ATTRIBUTE specified
SC--072	Not expecting more operand data
SC--073	Invalid subscript for set symbol
SC--074	Expecting an equal sign
SC--075	Invalid value at %s(0) %n for auto increment
SC--076	Subscript index must be between -1000000 and 1000000
SC--077	Invalid control word parameter
SC--078	Too many operands for .UD command
SC--079	Number of delimiters in running title %n %s exceeds four
SC--080	Expecting a valid numeric control word parameter
SC--081	Only one parameter allowed for this control word
SC--082	Value cannot be negative
SC--083	.HM + .HS must be less than or equal to .TM
SC--084	.FM + .FS must be less than or equal to .BM
SC--085	Control word parameter must be between 1 and 32
SC--086	The '%s' attribute was not completely defined
SC--087	For %s %n The symbol function is missing a right parenthesis
SC--088	For %s %n Must specify an additional numeric operand

SC--089	For %s %n Must specify an additional string operand
SC--090	For %s %n Expecting a valid numeric (integer) operand
SC--091	For %s %n The numeric operand must be greater than zero
SC--092	For %s %n Too many operands are specified
SC--093	For %s %n Invalid operand, expecting %s
SC--094	Space value is not valid
SC--095	.TM + .BM must be less than the page length
SC--096	Label name missing
SC--097	Label name too long
SC--098	Duplicate label '%s'
SC--099	Record number does not match '...%d'
SC--000	Invalid numeric .GO target, no line '%d'
SC--001	Missing .GO target
SC--002	Reached end of %s, forward %n .GO target '%s' not found
SC--003	Reached end of %s, unresolved %n .GO to line %d
SC--004	Can't find backward .GO target '%s'
SC--005	For .AP and .IM, line numbers can't be less than 1
SC--006	For .AP and .IM, ending line can't be less %n than starting line
SC--007	Extra operand(s) ignored: %n '%s'
SC--008	Environment stack exceeds 10 items
SC--009	Can't do .RE, environment '%s' %n does not exist
SC--010	Can't do .RE, environment stack is empty
SC--011	No active tab stops
SC--012	Tab position not found
SC--013	.LN can't be used inside .CC, .FN, .FK or other "in-storage" text block
SC--014	Line range is invalid for the .LN control word
SC--015	No further parameters expected after '%s' parameter
SC--016	String cannot be found from the change list for deletion
SC--017	Input record too large after change: %n %s
SC--018	Items have to be separated in Change Delete
SC--019	For .PU, workfile number must be between 1 and 9
SC--020	.ER %s: %s
SC--021	Numbered macro/remote labels must be %n between 1 and 32767
SC--022	Expecting DELETE, SAVE, NOSAVE or %n a remote invocation count
SC--023	Remote invocation count can't be %n less than 1
SC--024	Expecting SAVE or NOSAVE
SC--025	The resulting margin is too %s
SC--026	For %s, offset will go past current %n %s margin
SC--027	Not expecting a control word operand
SC--028	For %s %n Length value must be between 0 and %d
SC--029	For %s %n Value cannot be negative if length is not specified
SC--030	For .LS, the resulting spacing is too %s
SC--031	For %s, the resulting %s length is out of range

SC--032	Number of columns must be from 1 through 9
SC--033	Maximum gutter length must be no less than the minumum gutter length
SC--034	Expecting BEGIN, END, ON, OFF or number of pages
SC--035	Must be inside a footnote or a footnote leader
SC--036	Leading value must not have a sign
SC--037	The resulting page number is out of range
SC--038	The resulting paragraph indent is out of range
SC--039	Pending output cannot be cleared while a keep is active
SC--040	The running heading is too large to fit on the page
SC--041	The running footing is too large to fit on the page
SC--042	A .%s off must correspond to a previous .%s on

SY ***Internal System Messages***

SY--001	Memory exhausted!!!!!!!!!!
SY--002	Internal GML processing error

ZZ ***Continued Messages***

ZZ--001	Banner width is too small
ZZ--002	Banner depth is too small
ZZ--003	Depth of banner(s) too large for a page
ZZ--004	For region with hoffset = %s, voffset = %s,%n and indent = %s
ZZ--005	Banner regions overlap
ZZ--006	Cannot extend two regions into each other
ZZ--007	Banner region exceeds banner depth
ZZ--008	Banner region exceeds banner width
ZZ--009	System message is '%s'
ZZ--010	Cannot open file
ZZ--011	%d is an invalid attribute classification
ZZ--012	Virtual page not found
ZZ--013	Syntax stack not empty
ZZ--014	Premature end of syntax stack
ZZ--015	Page count exceeded
ZZ--016	'%s'
ZZ--017	have not been specified

LI ***Libman Messages***

LI--001	%s %nis not a library file or a directory
LI--002	%s %nis incompatible with this version of the library manager

APPENDIX C WATCOM GENDEV Error Messages

Some error messages are 'continued' by other messages. The ZZ error class contains the continuation messages, with references to them noted after the text of the starting error message.

A number of special symbols are used to insert information known only when the error occurs. The are:

%s	a text string replaces the %s
%d	a number replaces the %n
%n	a new line is started on the screen
%t	a tab to the screen position indicated by the number following the %t (zero(0) means the number is supplied when the error occurs)

The error messages are:

AT	***Attribute Messages***
AT--001	Required attribute not found
AT--002	Missing attribute value
AT--003	Invalid attribute value
AT--004	Attribute already found
AT--005	Missing or invalid closing quote on attribute
AT--006	Missing required equals
AT--007	The attribute value cannot have more than one character
CL	***Command line messages***
CL--001	Invalid option in command line
CL--002	Source file already specified
CL--003	Missing or invalid command filename
CL--004	Missing delimiter specification
DF	***Device Function Messages***
DF--001	Unrecognized device function tag
DF--002	Expecting an ending quote
DF--003	Need at least one hexadecimal digit following $
DF--004	Not a valid character in a device function

DF--005	Commas must separate device function parameters
DF--006	Too many parameters or too many commas
DF--007	Expecting more parameters
DF--008	This tag at start of device function sequence is invalid
DF--009	This device function tag cannot be a parameter of any tag
DF--010	This parameter cannot be used in this tag
DF--011	Need a newline with an advance of 1
DF--012	At least one device font definition must be specified

IN	***Informative Messages***
IN--001	Processing file
IN--002	Processing complete

IO	***I/O Messages***
IO--001	Cannot open file
IO--002	Recursive use of an include file
IO--003	Error reading device/font library
IO--004	Output operation failed
IO--005	Error reading input file
IO--006	GENDEV interrupted by ATTN key
IO--007	The device directory or library%n ’%s’%n does not exist

SN	***Syntax Messages***
SN--001	Number is too large or contains invalid characters
SN--002	Missing or invalid filename
SN--003	Invalid placement of text
SN--004	Expecting :eevalvalue tag
SN--005	Type is too long
SN--006	Fontname is too long or is an empty string
SN--007	Fontname already exists
SN--008	Type cannot be an empty string in :fontpause
SN--009	Width value must be an integer number
SN--010	Tag not supported in this version
SN--011	Expecting :enewline tag
SN--012	Expecting :enewpage tag
SN--013	Expecting :ehtab tag
SN--014	Unexpected tag encountered
SN--015	Expecting :startvalue tag
SN--016	Expecting :estartvalue tag
SN--017	Expecting :eendvalue tag
SN--018	Expecting :efontswitch tag
SN--019	Fontname not specified in :devicefont

SN--020	Duplicate :fontpause type
SN--021	Must have a :defaultfont with font=0
SN--022	The fontpause in :devicefont has no match in :fontpause
SN--023	Invalid location for a TEXTPASS directive
SN--024	More than one TEXTPASS directive specified in :lineproc
SN--025	Expecting :estartword tag
SN--026	Expecting :eendword tag
SN--027	Expecting :edefaultfont tag
SN--028	Expecting :efirstword tag
SN--029	Both a TEXTPASS directive and a ULINEON or ULINEOFF found in a :lineproc
SN--030	Duplicate :fontswitch type
SN--031	More than one :htab specified
SN--032	More than one :newpage specified
SN--033	Expecting :newpage tag
SN--034	Expecting text before termination tag.
SN--035	No text in input file
SN--036	Expecting :edriver tag
SN--037	Expecting :eintrans tag
SN--038	Expecting :efont tag
SN--039	Default Font already specified
SN--040	Expecting :value tag
SN--041	Expecting :epause tag
SN--042	Expecting :efontpause tag
SN--043	Expecting :edevice tag
SN--044	Pause already specified
SN--045	Expecting :einit tag
SN--046	Expecting :evalue tag
SN--047	Expecting :efontvalue tag
SN--048	Expecting :efinish tag
SN--049	Missing width value
SN--050	CMT tag should appear at start of line
SN--051	Invalid location for a ULINEON directive
SN--052	Invalid location for a ULINEOFF directive
SN--053	Duplicate :phrase exception chars
SN--054	Expecting :box tag
SN--055	Expecting :value tag
SN--056	except_chars value is too long
SN--057	Expecting :ebox tag
SN--058	Expecting :edevicefont tag
SN--059	For the defined name ’%s’%n the member name ’%s’ has already been used
SN--060	Expecting a ULINEON directive before a ULINEOFF
SN--061	No corresponding ULINEOFF directive for a ULINEON
SN--062	Expecting :eunderscore tag

SN--063	Expecting :epagestart tag
SN--064	Expecting :epageaddress tag
SN--065	Expecting :eabsoluteaddress tag
SN--066	More than one :pageaddress specified
SN--067	More than one :absoluteaddress specified
SN--068	Expecting :eouttrans tag
SN--069	Expecting :ewidth tag
SN--070	Missing input translation value
SN--071	Missing output translation value
SN--072	Expecting :ehline tag
SN--073	Expecting :evline tag
SN--074	Expecting :edbox tag
SN--075	More than one :hline specified
SN--076	More than one :vline specified
SN--077	More than one :dbox specified
SN--078	The line_height attribute must be specified %n if the scale_basis
SN--079	Expecting :epageoffset tag
SN--080	The font_space attribute must be specified %n if the font_height attribute is present
SN--081	Header at start of library file is invalid
SN--082	Current disk location and library path do not match
SN--083	Expecting :efontstyle tag
SN--084	Duplicate :fontstyle type
SN--085	Expecting :elineproc tag
SN--086	Invalid pass number
SN--087	2 lineproc blocks with identical pass numbers
SN--088	:fontswitch and :fontstyle have identical type
SN--089	Expecting :endvalue tag
SN--090	Expecting :ephrase tag
SN--091	Expecting :ebreakvalue tag
SN--092	Expecting :eoverride tag
SN--093	For font style '%s' %n the override value %s %n has not been declared
SN--094	Font style name may not contain spaces
SN--095	For the member name '%s' the defined name %n '%s' has already been used

SY ***Internal System Messages***

SY--001	Memory exhausted!!!!!!!!!!!
SY--002	Cannot find device or font in order to delete it
SY--003	Internal GENDEV processing error

LI ***Libman Messages***

LI--001	%s %nis not a library file or a directory
LI--002	%s %nis incompatible with this version of the library manager

Trademarks

Apple and LaserWriter are registered trademarks of Apple Computer, Inc.
DEC, VAX and VMS are registered trademarks of Digital Equipment Corporation.
HP is a registered trademark of the Hewlett-Packard Company.
IBM, IBM PC and IBM 370 VM/SP CMS are registered trademarks of International Business Machines Corporation.
Multiwriter V is a registered trademark of Ahearn & Soper Inc.
PostScript is a trademark of Adobe Systems, Inc.

A

B

E

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y