SH20-9160-0

Program Product

Document Composition Facility: Generalized Markup Language (GML) User's Guide

IBM

SH20-9160-0

Program Product

Document Composition Facility: Generalized Markup Language (GML) User's Guide

Program Number 5748-XX9

IBM

First Edition (July 1978).

This edition applies to Release 1 of the Document Composition Facility, program number 5748-XX9, and to all subsequent releases unless otherwise indicated in new editions or technical newsletters.

Information in this publication is subject to significant change. Any such changes will be published in new editions or technical newsletters. Before using this publication, consult the latest IBM System/370 Bibliography, GC20-0001, and the technical newsletters that amend the bibliography, to learn which editions and technical newsletters are applicable and current.

Requests for copies of IBM publications should be made to the IBM branch office that serves you.

Forms for readers' comments are provided at the back of this publication. If the forms have been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California 95150. All comments and suggestions become the property of IBM.

The Document Composition Facility, program number 5748-XX9, is a text processing system that supports the use of computers in the preparation of printed materials. The principal component of the Document Composition Facility is called "SCRIPT/VS." SCRIPT/VS provides facilities for manuscript preparation, text markup, page makeup and composition, and printing. It also provides a number of other document processing functions, among which is the ability to create your own markup interface, using the Generalized Markup Language (GML). GML provides the syntax and usage rules for developing your own vocabulary of "tags" for describing the parts of a document, without respect to particular processing.

This manual will show you how to define GML support for your installation. As an illustration of how to use SCRIPT/VS to develop GML support, IBM has created a "starter set" of GML. This manual also describes the starter set, and provides basic information on using SCRIPT/VS to process GML documents.

Note: The starter set of GML tags, profiles, and application processing functions (APFs) described in this manual are provided on an "as is" basis to expedite initial use of the Document Composition Facility. No warranty, express or implied, is provided for the starter set. Service support of the starter set is the responsibility of the customer.

Information about the use of batch processing and the Document Library Facility is for planning purposes only until availability of the Document Library Facility program product, program number 5748-XXE.

HOW TO USE THIS PUBLICATION

This manual addresses two distinct audiences: persons who define document production standards and procedures (called "document administrators" in this publication), and persons who mark up documents for processing in accordance with those standards and procedures (called "markup editors").

For document administrators, Parts One and Two of this manual are examples of how to prepare markup and operation guides for the types of document processed at your installation, and for the operating procedures you actually use. As such, they are written to be read by markup editors, and by persons producing their own documents informally.

Part One explains what GML markup is and how to mark up the type of document which the starter set supports. It includes detailed descriptions of some 50 tags and APFs provided in the starter set. Part Two shows how to invoke SCRIPT/VS, with examples from a number of system environments.

Part Three is addressed to persons performing the functions of a document administrator. It describes some of the decisions made in designing the starter set, and shows how to use GML to create markup tailored to the types of document produced at your installation.

Persons producing documentation informally, and users of the GML implementation known as "EasySCRIPT," may find "Appendix G: Getting Started with GML" on page 151 helpful. It explains how you can quickly put SCRIPT/VS to work on most text by using just a handful of starter set GML tags.

RELATED PUBLICATIONS:

- IBM Document Composition Facility: General Information Manual, GH20-9158, contains general Document Composition Facility product and planning information.
- IBM Document Composition Facility: User's Guide, SH20-9161, describes all SCRIPT/VS control words in detail, SCRIPT/VS symbol and macro facilities, and how to use SCRIPT/VS with other text processors and user programs.

- IBM Document Library Facility: General Information, GH20-9163, describes the Document Library Facility licensed program that must be installed to use SCRIPT/VS in batch processing environments.
- IBM Document Library Facility Guide contains specific information about the Document Library Facility functions available to batch SCRIPT/VS users.[1]
- Document Composition Facility: Generalized Markup Language (GML) - Quick Reference, SX26-3719, summarizes the starter set of GML tags, markup procedures, and processing options in a convenient reference card format.

REFERENCED PUBLICATIONS:

- IBM Virtual Machine Facility/370: CMS User's Guide, GC20-1819, gives detailed information about CMS text entry, editing, and file management.
- IBM OS/VS2 TSO Terminal User's Guide, GC28-0645, gives detailed information about TSO text entry, editing, TSO data set naming conventions, and TSO data set management.

[1] This publication will be available when the Document Library facility is available.

PART ONE: GML MARKUP GUIDE FOR GENERAL DOCUMENTS

This section explains what GML markup is, and the rules and conventions for entering it in a document. A type of document called a "General Document" is introduced, and 50 tags for markup of General Documents are described in detail. Special processing considerations for General Documents are also discussed. A summary of the material is included for quick reference in "Appendix A: GML Markup Summary" on page 131.

WHAT IS GML MARKUP?

You may be accustomed to typing documents on typewriters, where they appear exactly the way you typed them. Or, you may have worked with word processing or text processing systems where you "marked up" the text by entering special codes to achieve a desired output appearance. In both cases, you had to be concerned with such things as the length of lines and pages, hyphenation of words, alignment of columns, indentions, tab settings, and similar formatting considerations.

Markup with GML is different -- and very easy! You look over a document to determine the parts, or "elements," of which it is made up, such as paragraphs, figures, lists, and headings. You insert "element tags" to show where the elements begin and end, and what type of element it is. For example, ":fig" starts a figure and ":efig" ends it.

Not all element types require an explicit ending tag. The tag ":title" starts the title of the document; the title terminates at the end of the input line. (The individual tag descriptions, presented later in this manual, indicate when an explicit ending is required.)

You also insert "attribute labels" to give other information about the element, such as the depth of a figure or a shortened version of a heading.

```
:fig depth='2i'.
:efig
 .
 .
:h1 stitle='GML Guide'.GML Markup Guide for General Documents
```

The document into which you enter the text and markup is called the "source document"; it is created using the facilities available at your installation for text entry and editing.

The reason you do not have to put formatting codes in the source document is that a text programmer has already written "application processing functions (APFs)" for each type of element. For example, the APF for a list causes the text to be indented. The APF for an item of an ordered list causes the item to be numbered automatically. The formatting instructions, called "control words," are in the APFs, not in your document. That is why you do not have to be concerned with formatting when you mark up documents with GML.

Once the source document is prepared, you call the text processing program "SCRIPT/VS" to process it (using the command described later in this book). SCRIPT/VS produces a formatted "output document."

Because of GML, you can have the output document formatted in several different ways without changing your markup. This is done with other documents called "profiles," which have also been prepared by the text programmers. When you call SCRIPT/VS, you can indicate a specific profile, so it will know which APFs to associate with the tags and attribute labels you have entered. SCRIPT/VS will then perform all of the hyphenation, justification, page numbering, page layout, spelling verification, table of contents preparation, and other processing that may be required.

When you mark up a document, you must know what type of document it is, so that you can use the correct markup procedures for that type. For example, if your installation produces specialized types of documents, such as Directories, Procedure Manuals, or Standard Contracts, separate sets of tags and markup practices may have been created for each of them. If your document does not fit into a specialized category, its document type may be "General Document." This Markup Guide describes the tags and markup practices for General Documents.

IDENTIFYING DOCUMENT ELEMENTS

You might think of the text of a document as just a string of characters or words. Actually, documents have a natural structure, which in some cases can be fairly elaborate. This section explains the structure of General Documents so you will know how to mark them up.

The smallest things in a document, of course, are individual characters. Characters are combined to form words. Groups of words are phrases, which are combined to form sentences. Sentences themselves are elements of still larger structures, such as paragraphs.

Paragraphs, and other elements that have the same structure as a paragraph (described later), are called "paragraph units." Most of the time, elements that occur within a paragraph unit can clearly be identified in the text, and you will rarely need to mark them up with tags. For example, you can tell the bounds of a word by the interword blanks, and the end of a sentence by the punctuation.[1]

In other cases, such as phrases which need to be emphasized (highlighted), or quotations, we do need to use tags. This is because these elements are of a different type from the other elements in the paragraph unit. We call tagged elements that occur within the bounds of a paragraph unit "text items." Figure 1 shows the structure of a paragraph unit and the types of element, including text items, that can occur within it. (Of course, not every paragraph unit will have all of the elements shown in the figure. In fact, a paragraph unit could conceivably contain as little as a single word, or even a single character.)

```
paragraph unit (P, NOTE, or PC)
    sentence (no tag)
        figure reference (FIGREF)
        footnote reference (FNREF)
        heading reference (HDREF)
        highlighted phrase (HP0, HP1, HP2, HP3)
        phrase (PH)
        quote (Q)
            word (no tag)
                character (no tag)
```

Figure 1. Contents of a Paragraph Unit: Tags are in parentheses after the element type. Only paragraph units and text items have tags.

Normally, though, we use tags only to identify paragraph units, and higher-level elements that are made up of paragraph units. One group of elements for which you will need tags is called "basic document elements," because of the frequency with which they occur. Their tags are the ones you will use most often.

Some basic document elements are paragraph units. Others are made up of other tagged elements which themselves may be paragraph units, or may contain still other tagged elements. For example, an ordered list element contains list items and list parts. A list item contains a paragraph, but it can also contain most other

[1] Using natural text characters (such as a period at the end of a sentence) to mark up a document is called implicit markup. Using tags is called explicit markup.

basic document elements -- including lists. Such "nested" structures occur quite frequently. They are easy to mark up with the GML starter set, since you can always use the same tags, regardless of the level of nesting.

The basic document elements are listed in Figure 2, together with an indication of the types of element each can contain. The content of some elements is shown as "Text on Same Line." For such elements, you can enter only as much text as can fit on the same line as the tag. The text cannot include any tagged elements.

For other elements, the content is given as "No Immediate Text." This means you would not enter text immediately after the tag. Instead, you would enter the markup for another element which that element can contain. For example, long quotations can contain any basic document element except figures (**fig**).

For other elements listed in the figure, the content is shown as "Implied P Structure." Such elements always begin with an "implied" paragraph; that is, you do not enter a paragraph tag. The existence of the paragraph is understood from the existence of the tag for the implied paragraph structure. After the initial paragraph, an implied paragraph structure can also contain other basic document elements. (There are some restrictions, depending on the element type.)

You can find detailed explanations of all element types, where they can occur, and what they can contain, listed alphabetically by tag in the section "Element Tag Descriptions" on page 21.

Paragraph Units:

- note (NOTE)
- paragraph (P)
- paragraph continuation (PC)

Lists:

- definition list (DL)
  - definition term (DT): Text on Same Line
  - definition description (DD): Implied P Structure
  - list part (LP): Implied P Structure
- ordered list (OL)
  - list item (LI): Implied P Structure
  - list part (LP): Implied P Structure
- simple list (SL)
  - list item (LI): Implied P Structure
  - list part (LP): Implied P Structure
- unordered list (UL)
  - list item (LI): Implied P Structure
  - list part (LP): Implied P Structure

Other Basic Document Elements:

- address (ADDRESS)
  - address line (ALINE): Text on Same Line
- example (XMP): No Immediate Text
- figure (FIG)
  - figure body (no tag): No Immediate Text
  - figure caption (FIGCAP): Text on Same Line
  - figure description (FIGDESC): Implied P Structure
- long quotation (LQ): No Immediate Text

Figure 2. The 11 Basic Document Elements: Tags are in parentheses after the element type. Content is described after the tags.

The remaining elements of a General Document are shown in Figure 3 on page 7, which illustrates the overall structure. It shows that a General Document consists of four major elements: front matter, body, appendix, and back matter.

The front matter contains a series of elements of specialized types (title page, abstract, etc.) called "front matter segments." They do not occur anywhere else in the document.

The body, appendix, and back matter contain "heading segments." Heading segments are elements which begin with a heading, followed by basic document elements, and other heading segments. There are no tags for heading segments, since the tag for the heading will also identify the segment.

Heading tags are numbered to show the level of segment they begin: **h0** for "zero-level heading segment," **h1** for "first-level," and so on to **h6**. (A zero-level heading segment is called a "part." A first-level segment, when it occurs within the bounds of the body, is known as a "chapter.") When the document is printed, each level of heading will normally be formatted in a different style, to emphasize the structure of the text.

You will find that the more familiar you become with the structure of General Documents, the easier it will be to remember the tags and the places where you can enter them in a document. Figure 18 on page 133 summarizes all of the starter set tags in a way that illustrates the structure of General Documents.

```
•   Overall Structure:
        General Document (GDOC)
            front matter (FRONTM)
                title page (TITLEP)
                    document title (TITLE)
                    document number (DOCNUM)
                    document date (DATE)
                    author name (AUTHOR)
                    address of author or publisher (ADDRESS)
                        address line (ALINE)
                abstract (ABSTRACT)
                preface (PREFACE)
                table of contents (TOC)
                list of illustrations (FIGLIST)
            body (BODY): May contain either parts or chapters
                part (implied by H0)
                    head zero (H0)
                    basic document elements
                    chapter (implied by H1)
                        head one (H1)
                        basic document elements
                        lower-level heading segments (implied by H2 to H6)
            appendix section (APPENDIX)
                first-level heading segment (implied by H1)
            back matter (BACKM)
                first-level heading segment (implied by H1)

•   Other Heading Segments:
        nth-level heading segment (implied by H2 to H6)
            head n (H2, H3, H4, H5, H6)
            basic document elements
            lower-level heading segments

•   Special Elements:
        footnote (FN)
        process-specific controls (PSC)
```

Figure 3. Element Relationships in a General Document: Tags are in parentheses after the element type. Not all content is shown; consult tag descriptions for details.

MARKUP EXAMPLES

This section gives examples of markup and processing for some of the elements of a General Document.

GENERAL

Figure 4 on page 8 illustrates how starter set GML tags were used in marking up parts of this book. The formatting results of using the starter set tags depends on the APFs associated with them, the profile, and the output device.

```
:h1 id='mupro'.Markup Procedures
:p.This section explains:
:ul
:li.How to enter GML markup and text.
:li.Use of SCRIPT/VS symbols.
:li.When and how to use SCRIPT/VS control words.
:eul
:h2.Entering GML Markup and Text
:p.The rules for GML markup are:
:ol
:li.A document is marked up by identifying its elements.
It is almost always necessary to mark the start of an element
explicitly.
This is done with a tag preceded by a special
character that is used as the GML delimiter (&gml.).
Some examples are: &gml.fig, &gml.gdoc, &gml.title, and &gml.h1..
:p.Sometimes it is also necessary to mark the end of an element
explicitly.
This is done with the same tag used at the start, but
preceded by the GML
delimiter and an :q.e:eq. (for :q.end:eq.).
Some examples are &gml.efig and &gml.egdoc.
:p.Begin a new input line for all tags except text items.
:note.Miskeyed tags may be treated as text, or they may be mistaken for
other tags.
:li.Attributes are marked up in the form
:xmp
attribute-label='value'
:exmp
:pc.with at least one blank before the attribute label.
This markup is needed for each attribute.
The text or number that is the value of the attribute must be enclosed
by single quote marks (apostrophes), as shown.
If a single quote is used within the text of the value,
you should double it, so SCRIPT/VS will know it is not the end of the
value.
:p.For some attributes, the value is an amount of space, expressed
in :q.space units:eq..
Recommended space units are inches (nnI),
millimeters (nnW), picas/points (nnPnn), or
Ciceros/Didot points (nnCnn),
where :q.nn:eq. is the number of units.
```

Figure 4. Markup Example: The source for the start of the section "Markup Procedures" on page 11.

LISTS

There are several types of list element: definition, ordered, unordered, and simple. They can be nested (that is, occur within the bounds of one another), but within the bounds of a single list the level of nesting cannot be more than 3 for unordered lists, 6 for definition lists, 6 for simple lists, and 6 for ordered lists.

Simple lists (**sl**).

This is simple list item 1.

This is simple list item 2.

Unordered lists (**ul**), sometimes known as "bulleted" lists.

- This is unordered list level 1, item 1.
  - This is unordered list level 2, item 1.
    - This is unordered list level 3, item 1.
- This is unordered list level 1, item 2.

Ordered lists (**ol**), sometimes known as "numbered" lists.

1. This is ordered list level 1, item 1. Note that list items are automatically numbered or lettered when the document is processed; you need not enter the numbers. When an item is added or removed, renumbering is done for you.
   a. This is ordered list level 2, item 1.
      1) This is ordered list level 3, item 1.
         a) This is ordered list level 4, item 1.
         b) This is ordered list level 4, item 2.
      2) This is ordered list level 3, item 2.
   b. This is ordered list level 2, item 2.
2. This is ordered list level 1, item 2.

Definition lists (**dl**), which contain definition terms (**dt**) and corresponding definition descriptions (**dd**).

Term1	Definition description for term one.
Term2	Definition description for term two.

HIGHLIGHTING

Certain elements of a document may be emphasized, or "highlighted," which usually requires that they contrast with unemphasized text when printed. Four kinds of highlighting are possible with the starter set. They are: Highlight 0, _Highlight 1_, **Highlight 2**, and **_Highlight 3_**.[2] In the starter set, Highlight 0 is set like normal text. **It can be used for a** deemphasized element **that occurs within an emphasized element.**

In a General Document, the elements which can be highlighted are the phrase (**ph**), the four kinds of highlighted phrase (**hp0**, **hp1**, **hp2**, and **hp3**), and the definition term (**dt**). Highlighting for the phrase is specified by the **hi** attribute label. Highlighting for the definition term is specified by the **termhi** attribute of the definition list (**dl**).

CROSS-REFERENCES

Footnotes, figures, and certain heading elements (**h0** through **h4**) may have an attribute called a "unique identifier." The value of the unique identifier is a string of 1 to 5 alphameric characters which serves as a name which can be used to refer to the element. Two footnotes, two figures, or two headings (of any type) cannot have the same unique identifier. The unique identifier is specified by the **id** attribute label.

[2] Wherever the starter set uses bold face for 3800 output, upper case is used instead for single-font devices.

```
:fn id='impmu'.Using natural text characters . . .
(remaining footnote text)
:efn
  .
  .
:fig id='btu'
(body of figure)
:figcap.The 11 Basic Document Elements
:figdesc.Tags are in parentheses after the element type.
Content is described after the tags.
:efig
  .
  .
:h3 id='me'.Markup Examples
```

There are three other elements which are references to the footnote, figure, and heading elements. They are footnote reference (**fnref**), figure reference (**figref**), and heading reference (**hdref**), respectively. Each has an attribute called "reference identifier " which is specified by the **refid** attribute label. Its value is the unique identifier of the element being referred to.

```
...implicit or not.:fnref refid='impmu'
...see :figref refid='btu'. for details.
...discussed in :hdref refid='me'..
```

In the starter set, the content of a reference element is generated by its APF. Depending upon the element, the content will consist of a footnote number, a figure number, or the text of a heading. For figure and heading references, the generated content will include the page number if the element being referred to is not on the same page as the reference to it.

```
...implicit or not.¹
...see Figure 2 on page 6 for details.
...discussed in "Markup Examples" on page 7.
```

This section explains:

- How to enter GML markup and text.
- Use of SCRIPT/VS symbols.
- When and how to use SCRIPT/VS control words.

ENTERING GML MARKUP AND TEXT

The rules for GML markup are:

1. A document is marked up by identifying its elements. It is almost always necessary to mark the start of an element explicitly. This is done with a tag preceded by a special character that is used as the GML delimiter (:). Some examples are: :fig, :gdoc, :title, and :h1.

   Sometimes it is also necessary to mark the end of an element explicitly. This is done with the same tag used at the start, but preceded by the GML delimiter and an "e" (for "end"). Some examples are :efig and :egdoc.

   Begin a new input line for all tags except text items.

   Note: Miskeyed tags may be treated as text, or they may be mistaken for other tags.

2. Attributes are marked up in the form

   ```
   attribute-label='value'
   ```

   with at least one blank before the attribute label. This markup is needed for each attribute. The text or number that is the value of the attribute must be enclosed by single quote marks (apostrophes), as shown. If a single quote is used within the text of the value, you should double it, so SCRIPT/VS will know it is not the end of the value.

   For some attributes, the value is an amount of space, expressed in "space units." Recommended space units are inches (nnI), millimeters (nnW), picas/points (nnPnn), or Ciceros/Didot points (nnCnn), where "nn" is the number of units. Inch and millimeter units may be expressed as a decimal fraction with one or two decimal places.

   Each attribute label and its corresponding value must be on the same input line as the element tag. If there is insufficient space, you can use the technique for attribute insertion described in the section on "Inserting Attributes (.ATT)" on page 14.

3. If an element has text, the text must begin on the same line as the markup. Markup is the element tag and any attribute labels and their values.

   Note: For some element types, all of the text must be on the same line as the markup. These are described in the individual tag descriptions.

4. Whenever markup is followed by text on the same input line, the markup must be separated from the following text by a period (.). There should be no blanks before the period. Blanks after it are treated as part of the text.

   Note: This rule applies to markup for the end of an element as well as markup for the start.

5. Start all lines at the left margin. There should be no leading blanks. (Trailing blanks at the end of an input line are ignored.)

6. When you end a sentence, do not enter any more text on that line.

7. Do not begin a line with a period unless you are entering a SCRIPT/VS control word. (Control words are identified by a period in the first position of an input line.)

The following practices are also recommended:

1. Do not mark an element with a tag that does not correctly describe it, even if it results in the correct processing. If a document has elements of a type not described in this manual, consult your document administrator.

2. Enter tags entirely in lower case, as shown in the individual tag descriptions.

3. Do not enter any blank lines. (This applies to "null" lines and "index returns" as well, for those keyboards which ordinarily permit them.)

4. Keep your input lines short -- no longer than approximately 70 characters. Short lines make revisions easier. However, a line may be as long as 132 characters, if needed.

5. Do not underscore or overstrike any characters.

6. Do not use the tab key.

7. For convenience in handling, you may divide a document into parts and store each part in a separate computer file. Make sure that each file starts at the beginning of a paragraph unit or higher level element, and ends at the end of one.

Note: Some of the foregoing practices may not apply when entering text and control words as part of a **psc** element (explained later), an example, or a figure.

SCRIPT/VS SYMBOLS

A symbol is a string of characters that begins with an ampersand (&) and ends with a period (.). When you include a symbol in a source document, SCRIPT/VS will replace it with other characters in the output document. Your installation may have defined symbols for such purposes as:

- Entering characters that are not on the input keyboard.
- Abbreviating frequently-used lengthy phrases.
- Avoiding confusion with characters that have a special meaning to SCRIPT/VS.

A symbol may be entered anywhere in a document where the characters the symbol represents could have been entered. One exception to this rule is, GML tags cannot contain symbols. (You can, however, use a symbol within the value of an attribute.) The starter set includes the following symbols:

&amp.	This symbol generates an ampersand (&). Use it instead of the ampersand whenever it is immediately followed by a letter, a number, or the characters @, $, or #. This will prevent the text from being interpreted as a symbol.
&gml.	GML delimiter (:). When your document contains examples of GML markup, use this symbol in the examples instead of the real GML delimiter character. This prevents the markup examples from being interpreted as real markup.
&rbl.	Required blank. When you wish adjacent words to be treated as a single word for the purpose of checking spelling, or to keep them on the same output line as one another, use &rbl. to separate them, instead of the ordinary blank.

If you miskey a symbol, it will be treated as text, unless it is mistaken for another symbol.

Note: Your document administrator may be able to advise you of other symbols you may find useful in your documents.

CONTROL WORDS AND MACROS

There are a number of situations in which it will be necessary to supplement GML markup with direct entry of control words or macros.[3] These are:

- to use SCRIPT/VS for source document management;
- to obtain graphic formatting not provided by available APFs;
- to modify processing results; and
- to supplement the general processing specified in the profile.

The control words described in this section must be entered at the left margin, preceded by a period (.). Some control words require you to specify additional items of data, called "parameters." They are entered on the same line as the control word, preceded by a blank. If there are several parameters, they must be separated by blanks.

Note: You should not enter control words except in accordance with the procedures in this section. Although acceptable processing output may be produced, the control words may make it more difficult to maintain the document and to use it for other applications. You can follow these procedures and still obtain all of the functions which direct entry of control words can provide.

SOURCE DOCUMENT MANAGEMENT

You may want to store parts of your document as separate files and have SCRIPT/VS combine them for processing. You may want to write comments to readers of your source document, or identify revised portions of the text. Or you may need several input lines for attributes which SCRIPT/VS expects to find on one line. The following control words will help you handle these situations.

Imbedding Files (.IM)

This control word causes the file whose file-identifier is given as the parameter to be processed at this point, as though the current file and the imbedded file were one single file. Many different files can be imbedded to form a single document.

For example:

```
.im chap2
```

Writing Comments (.CM)

To write a comment to readers of the source document (such as yourself), enter:

```
.cm text of comment -- if too long for one line, then
.cm continue it like this.
```

Indicating Revisions

You can call attention to a revised portion of a document by causing it to be flagged. Consult your document administrator for your installation's procedure for doing this.

[3] A macro is a processing instruction which is created by a text programmer. The rules and conventions for entering control words also apply to macros.

Inserting Attributes (.ATT)

Sometimes you may not have enough room on one line to mark up all of an element's attributes. The attribute insertion macro control (.ATT) allows for additional attribute labels on the lines above the related document element. SCRIPT/VS inserts them in the correct place. (This is the way a caret might be used on a manuscript, to indicate that additional text, written above a line, is to be inserted.)

```
    .att stitle='Purchasing List'
    :h1.Acme Company Consolidated Interdivisional Purchasing List
```

You may have more than one attribute label and corresponding value with one .ATT macro control. Also, you may have more than one .ATT macro control above an element. In each case, the attribute label and value are treated as if inserted ahead of the first blank or period on the following line.

```
    .att id='purl'
    .att stitle='Purchasing List'
    :h1.Acme Company Consolidated Interdivisional Purchasing List
```

GRAPHIC FORMATTING

Your document may require formatting which is not conveniently available with the APFs provided by your installation. For example, you may wish to use character graphics to create an illustration, such as a flow chart, or a complex table. Illustrations can be done with specific markup. However, different versions of the marked-up illustration may be required for different processing that may be performed on the document. For example, producing a single-column draft on a 2741 terminal would be a different process from producing a two-column final copy on the 3800 printer.[4]

To make this flexibility possible, you should enter such formatting control words only within a special element called a "process-specific controls" element (**psc**). An attribute of the **psc** element lets you specify the processes to which it applies. The first line of the element should be a comment that indicates that the element is being used for permanent graphic formatting.

When a **psc** element is used in this way it is called a "graphic." As the following example illustrates, you can have more than one version of a graphic. SCRIPT/VS automatically includes the version for the process being run, and ignores the others.

```
    :fig
    :psc proc='1403 TERM'
    .cm graphic
    (formatting control words and data)
    :epsc
    :psc proc='3800'
    .cm graphic
    (other formatting control words and data)
    :epsc
    :figcap.Typical Diagram
    :efig
```

To mark up a graphic, you must know how to use the specific markup language. For further details, including the list of acceptable processes, see the description of "psc -- process-specific controls" on page 72.

MODIFYING PROCESSING RESULTS

You may occasionally be dissatisfied with the results produced by your formatter, whether it be SCRIPT/VS, or some other formatter for which SCRIPT/VS is being used as a preprocessor. For example, the formatter might choose to break a column or page at a bad point; or, it might leave unwanted white space on the page. You would

[4] IBM 3800 Printing Subsystem

not want to make a permanent change in the document markup to cure such problems, because they might disappear when the text of the document is revised. Likewise, they might only occur on one output device, and not on others.

The situation is similar to those that require graphic **psc** elements. In both cases, you must tell the system which processes apply. However, there is one significant difference. The graphic element is a permanent part of the document, while the modification is only temporary.

For this reason, although the **psc** element tag is used, the following comment line should identify the element as a temporary modification, or "patch."

```
:psc proc='TERM'
.cm patch
(formatting control words)
:epsc
```

A patch might include text interspersed with the control words. When this is the case, you might need two or more patches for the single situation that needs repair: one to correct the condition, and the other to make sure the text prints normally for the other output devices. The "normal" patch will usually have more than one process specified.

Note: Remember to remove from your document any patches which become unnecessary. Patches may become unnecessary when editorial revisions, or changes in the formatting APFs, have the effect of eliminating the undesirable output.

ADDITIONAL GENERAL PROCESSING

The profile you specify when you use SCRIPT/VS contains a number of control words that perform functions other than formatting. These include the associating of GML tags with APFs, and the setting of values of symbols. You can request additional functions of this nature for a particular document by including the desired control words within a **psc** element. A **psc** so used should include a comment indicating its use in supplementing the document profile.

```
:psc
.cm profile
(general processing control words)
:epsc
```

Where possible, **psc** profile elements should be placed at the beginning of a document. However, you may insert them elsewhere as necessary within a document. You should include detailed comments in the **psc** to explain the purpose of the control words.

In order to process your documents, you will need to know how to issue the command that will cause the computer to invoke SCRIPT/VS. You will also need to be familiar with some of the options which can be issued with the command, and the method of requesting them. This knowledge is applicable to the processing of all types of documents, and therefore is not discussed here. You can learn about it in "Processing GML Documents With SCRIPT/VS" on page 89. This section discusses processing considerations unique to General Documents.

CHOOSING THE CORRECT PROFILE

When a document is ready to be processed, you issue the "SCRIPT" command to the computer to invoke SCRIPT/VS. One of the options of the command permits you to name a profile, which tells SCRIPT/VS which APFs to associate with your markup. Different profiles cause different processing to occur. With some profiles you may be able to specify variations of the processing, using the "SYSVAR" option of the SCRIPT command.

Profile(s) available for use with General Documents are:

gdocprof Produces formatted output on terminals, line printers, and the IBM 3800 Printing Subsystem. Verifies spelling when the "SPELLCHK" command option is specified.

SYSVAR parameters for variations are:

c 1, c 2	Columns: Prints the body, appendix (if any), and back matter (if any) in the number of columns specified (1 or 2). If omitted, prints in double-column format on the 3800, and single-column format on other devices.
d yes	Duplex: Begins new odd-numbered page for **h0**, **h1**, and front matter segments. This is the preferred style for making master pages for two-sided ("duplex") reproduction. If omitted, begins new page which may be either even or odd.
h num	Head Numbering: Automatically numbers the heading types **h1**, **h2**, **h3**, and **h4**. If omitted, prints headings as entered, without numbering them.
p process-name	Process: Lets you specify the process being run. If omitted, the names of the physical output device and the logical device are used as active processes for **psc** elements. See "psc -- process-specific controls" on page 72 for details.
t no	Title Page Printing: Suppresses printing of the title page. If omitted, the title page is printed.

For example, the following command will cause output to be printed in two columns with numbered headings, not duplexed, and with a title page. Only **psc** elements with the process attribute of "TERM" or "2741" will be processed, because they are the default logical and physical devices.[5]

```
  SCRIPT mydoc ( PROFILE (gdocprof) SYSVAR (c 2 h num)
```

Variations other than the heading numbering and number of columns were defaults, and were not specified in the command.

Note: Consult your document administrator for other profiles and variations he may have established; such as for double-spaced drafts or for pre-processing documents for use with other text processors. He may also have prepared "execs," "procedures," or "option" files for specific tasks that make it unnecessary to

[5] The CMS syntax is used in the command. For TSO, the initial left parenthesis is omitted. The command word "SCRIPT" and the option names are shown in upper case for emphasis.

specify the individual profile and variations.

INTERIM PROCESSING

This manual explains how to mark up a General Document in its "normal" form. This is the form in which it is processed for final printing, and in which it remains in your computer files for any processing that may be required in the future. Profiles and APFs are primarily designed to work correctly when a document is in the normal form.

During the course of production, though, your document may not always be in the normal form. At times it will be incomplete, or incorrectly marked up. Or, you might wish to process only a part of the document -- perhaps a heading segment which was just revised. The starter set profile and APFs allow you considerable flexibility during the production cycle:

- You can temporarily enter the **toc** and **figlist** tags as the last tags in the back matter, thereby obtaining a table of contents and a list of illustrations without the expense of two-pass processing. (Note that you may want the second pass anyway, if you need to check the resolution of cross-references.)
- Parts of the document can be processed independently, although results may not be identical to those you will obtain when the entire document is processed in the normal form. (You can improve these interim results by including some of the higher-level tags in the processing run. For example, if you wish to process only a second-level heading segment, you may obtain better output if you insert the **gdoc**, **body**, and **h1** tags ahead of it, since their APFs will establish the proper page layout, running headings, heading numbering (if requested), and so on.) By experimenting, you can determine which element types you can satisfactorily process independently.
- Incomplete or inaccurate markup can cause errors which would normally persist for the remainder of the document. For example, failure to terminate a list might cause succeeding text to be indented. To minimize the impact of such errors, some APFs perform "housekeeping" functions in addition to their normal processing. For example, the heading APFs terminate any current lists and highlighting. (These safeguards will make it more likely that your draft copies will appear to be correct, but you should nonetheless correct the error conditions with the proper markup. Errors are discussed further in the next section.)

MARKUP ERRORS

There are two kinds of markup error:

1. entering markup you think is right, but isn't; and
2. making a typographical error that causes SCRIPT/VS to misinterpret markup (or to treat as markup something that is not).

You can prevent the first type of mistake by understanding this manual, and by taking care in identifying the elements of your document. The second type of mistake obviously can be avoided by careful typing.

Of course, no one is perfect, and errors will be made. The next best thing to avoiding errors is being able to recognize where they are, and to correct them without difficulty. In computer text processing, finding and correcting markup errors is called "debugging" your markup.

Some common markup "bugs" are:

- Omitting the period at the end of a symbol.
- Omitting the period at the end of markup (when required).
- Omitting the ending tag of a list, figure, example, or other element that requires explicit termination.

- Ending a quote (**q**) with a quotation mark instead of **eq**.
- Using tags where they are not permitted, such as within an element whose content is "Text on Same Line."
- Not entering the ending tag of a text item immediately after the last character of the element, and on the same input line. (This may cause an extra blank to be included in the text item.)
- Omitting the **p** tag for a paragraph that is not the start of an "Implied P Structure," such as a paragraph that follows a heading or figure.

For example, you might find that entire pages of your output are underscored. This would normally be caused by omitting an end of highlighted phrase tag. (You could actually have forgotten the tag, or you might just have forgotten the period at the end of it. In that case, SCRIPT/VS would not know it was a tag, and would treat it as text. Either way, the result is the same.) This bug is easy to find: just go to the point where the underscoring begins, and start looking for an incorrectly entered end of highlight tag (or the place where the highlighting should have ended.)

It is important to realize that even though dozens of words (or even dozens of pages) may look wrong in the output, it does not necessarily mean you must make dozens of corrections in the source. In fact, you will often find that the worse your output looks, the easier it will be to locate the bugs and correct them.

Sometimes an APF can determine that a markup error was made. In such cases, it will take some appropriate processing action, and issue a message to help you correct the error. The text of messages issued by the starter set APFs, together with explanations, can be found in "Appendix C: GML Messages" on page 141.

You may also receive error messages from SCRIPT/VS itself. You should consult your document administrator about these.

The following pages describe the elements of a General Document, how to mark them up, and the processing that is performed on them. Each element description includes a box that shows:

Type	The tag for the element type.
Attributes	The labels for the attributes (if any).
Content	If the markup is to be followed immediately by text, this will tell if it all must fit on the same input line as the markup ("Text on Same Line"), or whether there can be many lines of text ("Text Item," "Paragraph Unit," or "Implied P Structure"). If it says "No Immediate Text," nothing else should be entered on the line. The next line will begin with the tag for another element. If it says "Generated by APF," you should not enter any content for that element. However, if the element is a text item, you can continue to enter text on the same line.
Termination	How to mark up the end of the element. Elements whose content is "Text on Same Line" terminate at the end of the input line ("At End of Line"). Elements whose content is "Generated by APF" are self-terminating. That is, the end of the markup is the end of the element ("At End of Markup"). Some elements are terminated explicitly; for these, the ending tag will be shown.

The remaining elements are terminated by the tag for a new element which is at the same or a higher level in the document's structure. For example, a paragraph unit is terminated by a "Paragraph or Higher Level Element." Similarly, a chapter would be terminated by another chapter (same level) or the appendix section (higher level). A head 3, or a basic document element, though, would be at a lower level and would not terminate the chapter; it would be part of the chapter's content. The box will either show the actual tags that could terminate the element, and/or direct you to the text for an explanation ("See "Usage" in text").

When an element ends, any unterminated elements in its content are terminated at the same time.

Below the box are descriptions of the element and its attributes, followed by discussions under the following headings:

USAGE: This discusses where in a document's structure the element can occur, and what elements it can contain, together with any special entry rules.

Some elements, such as title page elements, must always occur in a specific place in the document. These elements are frequently optional (need not occur in every document) and/or repeatable (can occur more than once in succession). When this is the case, it is mentioned in this section.

PROCESSING: Markup is the same regardless of the profile you will be using. Processing, though, depends upon the profile, APFs, and output device, and may be varied by options of the SCRIPT command. For this reason, processing is discussed separately for each profile if more than one is available. (Only one -- GDOCPROF -- is supplied with the starter set.)

The processing examples in the tag descriptions will vary depending upon the way this manual itself was processed. Printers traditionally include in their books a statement, called a "colophon," which tells how the book was composed and printed. So you will know the conditions under which the processing examples were produced, SCRIPT/VS has automatically generated the following colophon.[6]

[6] The CMS syntax is used in the command. The profile is not GDOCPROF. It is another profile which includes GDOCPROF, plus other tags needed for markup guides. The processing examples are the same as if GDOCPROF were the profile.

This document was produced with SCRIPT/VS on a 3800 (logical device 3800N8) using the following command:

```
 SCRIPT GMLGUIDE ( PROFILE ( GMLGPROF ) SYSVAR ( D YES C 1 ) DEV ( 3800N8 )
     TWO BIND ( 7P6 3P6 ) FILE ( $GMLGUID )
```

Type	Attributes	Content	Termination
:abstract	None	No Immediate Text	:preface, :toc, :figlist, or :body

The **abstract** tag identifies a summary of the document.

Usage: The abstract can occur only within the front matter, after the title page. It can contain basic document elements and second-level and lower heading segments. This element is optional.

```
  .
  .
:frontm
:titlep
  .
  .
:etitlep
:abstract
:p.This manual describes a method by which you can
obtain the benefits of a powerful text processing system without
becoming an expert in composition.
  .
  .
:preface
  .
  .
:toc
:figlist
:body
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF begins a new page (odd-numbered if the SYSVAR D YES option is specified) and generates the word "Abstract" as a heading.

<u>Results:</u>

<u>See the preface of this manual for an example. It is formatted in the same way as an abstract, except that the word "Preface" is generated.</u>

Type	Attributes	Content	Termination
:address	None	No Immediate Text	:eaddress

The **address** tag identifies a street address or mailing address. Within the title page, it identifies the address of the author or publisher.

Usage: The address can occur within the title page, where it is optional and repeatable. It can also occur wherever a basic document element can occur. Its content is one or more address line (**aline**) elements which contain the actual text of the address.

```
  .
  .
:frontm
:titlep
:title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
:docnum.MG-00001
:date.July 3, 1978
:author.L. T. Smith, Document&rbl.Administrator
:address
:aline.Any Company
:aline.500 Main Street
:aline.Anycity, Anyplace
:eaddress
:etitlep
:preface
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF formats the address as shown below, when it is not entered as part of the title page. See "Appendix D: Title Page Example" on page 143 for placement and style of the address on the title page.

Results:

```
Any Company
500 Main Street
Anycity, Anyplace
```

ALINE -- ADDRESS LINE

Type	Attributes	Content	Termination
:aline	None	Text on Same Line (no other tags)	At End of Line

The **aline** tag identifies a line of an address.

Usage: The address line can occur only as an element of an address. It is repeatable.

```
      .
      .
    :frontm
    :titlep
    :title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
    :docnum.MG-00001
    :date.July 3, 1978
    :author.L. T. Smith, Document&rbl.Administrator
    :address
    :aline.Any Company
    :aline.500 Main Street
    :aline.Anycity, Anyplace
    :eaddress
    :etitlep
    :preface
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF formats the address lines as shown below, when they are not entered as part of the title page. See "Appendix D: Title Page Example" on page 143 for placement and style of the address lines on the title page.

Results:

```
    Any Company
    500 Main Street
    Anycity, Anyplace
```

APPENDIX -- APPENDIX SECTION

Type	Attributes	Content	Termination
:appendix	None	No Immediate Text	:backm or :egdoc

The **appendix** tag identifies that major element of a document which contains explanatory and illustrative material helpful to the reader, but not essential to the main text.

Usage: The appendix section can occur immediately after the body. It is optional. It contains one or more first-level heading segments.

```
    :gdoc sec='Company Confidential'
    :frontm
      .
      .
    :body
      .
      .
    :appendix
    :h1.GML Diagonostic Messages
      .
      .
    :backm
      .
      .
    :egdoc
```

Processing With the "gdocprof" Profile: The supplied APF causes the word "Appendix" and a serial letter to be printed as a prefix to each head one (**h1**) in the appendix section. For 3800 output, the appendix section is set in two columns unless the SYSVAR C 1 option is specified. For other devices, it is set as a single column unless the SYSVAR C 2 option is specified.

Results:

See "Appendix D: Title Page Example" on page 143 for an example.

AUTHOR -- AUTHOR NAME

Type	Attributes	Content	Termination
:author	None	Text on Same Line (no other tags)	At End of Line

The **author** tag identifies the writer of the document.

Usage: The author name can occur only within the title page. It is optional and repeatable. (That is, it may be entered more than once for more than one author.)

```
      .
      .
:frontm
:titlep
:title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
:docnum.MG-00001
:date.July 3, 1978
:author.L. T. Smith, Document&rbl.Administrator
:address
:aline.Any Company
:aline.500 Main Street
:aline.Anycity, Anyplace
:eaddress
:etitlep
:preface
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF prints the author name on the title page.

Results:

See "Appendix D: Title Page Example" on page 143 for an example.

BACKM -- BACK MATTER

Type	Attributes	Content	Termination
:backm	None	No Immediate Text	:egdoc

The **backm** tag identifies that major element of a document which includes such reference information as a glossary, a bibliography, and an index.

Usage: The back matter can occur at the end of the document, immediately after the appendix section (or after the body if there is no appendix section). The back matter is optional. It contains one or more first-level heading segments.

```
:gdoc sec='Company Confidential'
:frontm
  .
  .
:body
  .
  .
:appendix
  .
  .
:backm
:h1.Glossary
  .
  .
:h1.Index
  .
  .
:egdoc
```

Processing With the "gdocprof" Profile: The supplied APF suppresses automatic heading numbering (when it is requested) for headings within the bounds of the back matter. For 3800 output, the back matter is set in two columns unless the SYSVAR C 1 option is specified. For other devices, it is set as a single column unless the SYSVAR C 2 option is specified.

Results:

See the back matter of the IBM-published version of this manual for an example of the processing results.

BODY -- BODY

Type	Attributes	Content	Termination
:body	None	No Immediate Text	:appendix, :backm, or :egdoc

The **body** tag identifies that major element of a document which contains the main text.

Usage: The body can occur immediately after the front matter. It can contain either a series of parts (**h0**) or a series of chapters (**h1**).

```
    :gdoc sec='Company Confidential'
    :frontm
      .
      .
    :figlist
    :body
      .
      .
    :h1.What is GML Markup?
      .
      .
    :h1.Marking Up General Documents
      .
      .
    :appendix
      .
      .
    :backm
      .
      .
    :egdoc
```

Processing With the "gdocprof" Profile: The supplied APF causes body pages to be numbered in Arabic numerals. For 3800 output, the body is set in two columns unless the SYSVAR C 1 option is specified. For other devices, it is set as a single column unless the SYSVAR C 2 option is specified.

Results:

See the Arabic numeral pages of this manual for an example of the processing results.

DATE -- DOCUMENT DATE

Type	Attributes	Content	Termination
:date	None	Text on Same Line (no other tags)	At End of Line

The **date** tag identifies a date associated with the document, such as the date of creation, publication, or revision.

Usage: The document date can occur only within the title page.

```
  .
  .
:frontm
:titlep
:title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
:docnum.MG-00001
:date.July 3, 1978
:author.L. T. Smith, Document&rbl.Administrator
:address
:aline.Any Company
:aline.500 Main Street
:aline.Anycity, Anyplace
:eaddress
:etitlep
:preface
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF prints the contents of the date element on the title page. If the date tag is entered with no text, the title page will contain a system-supplied processing date.

Results:

See "Appendix D: Title Page Example" on page 143 for an example.

DD -- DEFINITION DESCRIPTION

Type	Attributes	Content	Termination
:dd	None	Implied P Structure	:dt, :lp, or :edl

The **dd** tag identifies a definition, description, or explanation of a word or phrase in a definition list.

Usage: The definition description can occur only within a definition list (**dl**), immediately after its corresponding definition term element (**dt**). There must be a **dd** for each **dt**. It contains an implied paragraph (no **p** tag), and may also contain basic document elements.

```
:p.This paragraph precedes the list.
:dl
:dt.Term1
:dd.The first definition description.
This is an implied paragraph -- there was no :hp2.p:ehp2. tag.
:p.This paragraph is part of the first definition description.
:dt.Term2
:dd.The second definition description.
:lp.This list part introduces succeeding terms and descriptions.
:p.A list part could comment on preceding terms and descriptions.
:dt.Term3
:dd.The third definition description.
It has two topics:
:ol
:li.one topic;
:li.another topic.
:eol
:dt.Term4
:dd.The fourth definition description.
:edl
:p.This paragraph follows the list.
```

Processing With the "gdocprof" Profile: The supplied APF formats the definition terms and definition descriptions as a hanging indent list, with the indention equal to the specified term size (10M if the **tsize** attribute of the definition list is not specified). The terms are assumed to have a highlighting attribute of 2 if the **termhi** of the definition list is not specified. (See "Highlighting" on page 9 for an explanation of highlighting.)

<u>Results:</u>

This paragraph precedes the list.

Term1	The first definition description. This is an implied paragraph -- there was no **p** tag. This paragraph is part of the first definition description.
Term2	The second definition description.

This list part introduces succeeding terms and descriptions.

A list part could comment on preceding terms and descriptions.

Term3	The third definition description. It has two topics: 1. one topic; 2. another topic.
Term4	The fourth definition description.

This paragraph follows the list.

DL -- DEFINITION LIST

Type	Attributes	Content	Termination
:dl	termhi= tsize=	No Immediate Text	:edl

The **dl** tag identifies a list of words and phrases and their corresponding definitions, descriptions, or explanations.

The **termhi** attribute label identifies the optional definition term highlight attribute. Its value is a number from 0 to 3 which identifies the type of emphasis associated with the definition terms in this list. (See "Highlighting" on page 9 for an explanation of highlighting.)

The **tsize** attribute label identifies the optional definition term size attribute. Its value is a number of "ems" (entered 'nnM') which is one greater than the number of characters of the longest definition term that could be in the list.

Usage: The definition list can occur wherever a basic document element can occur. It contains a series of definition list items, which may be intermixed with individual elements known as "list parts" (**lp**). Each definition list item is actually a word or phrase -- the "definition term" (**dt**) -- and its corresponding definition or explanation -- the "definition description" (**dd**).

All types of list (definition, ordered, unordered, and simple) can occur within the bounds of one another), but within the bounds of a single list the level of nesting cannot be more than 3 for unordered lists, 6 for definition lists, 6 for simple lists, and 6 for ordered lists.

```
:p.This paragraph precedes the list.
:dl tsize='6M' termhi='3'
:dt.Term1
:dd.The first definition description.
This is an implied paragraph -- there was no :hp2.p:ehp2. tag.
:p.This paragraph is part of the first definition description.
:dt.Term2
:dd.The second definition description.
:lp.This list part introduces succeeding terms and descriptions.
:p.A list part could comment on preceding terms and descriptions.
:dt.Term3
:dd.The third definition description.
It has two topics:
:ol
:li.one topic;
:li.another topic.
:eol
:dt.Term4
:dd.The fourth definition description.
:edl
:p.This paragraph follows the list.
```

Processing With the "gdocprof" Profile: The supplied APF indents the list appropriately to its level of nesting. Terms and descriptions are formatted as a hanging indent list, with the indention equal to the specified term size (10M if **tsize** is not specified). If an actual term is longer than the specified size, the term will be separated from the start of the definition description text by a single interword space. The terms are assumed to have a highlighting attribute of 2 if **termhi** is not specified.

Results:

This paragraph precedes the list.

Term1 The first definition description. This is an implied paragraph -- there was no **p** tag.

This paragraph is part of the first definition description.

Term2 The second definition description.

This list part introduces succeeding terms and descriptions.

A list part could comment on preceding terms and descriptions.

Term3 The third definition description. It has two topics:

1. one topic;
2. another topic.

Term4 The fourth definition description.

This paragraph follows the list.

DOCNUM -- DOCUMENT NUMBER

Type	Attributes	Content	Termination
:docnum	None	Text on Same Line (no other tags)	At End of Line

The **docnum** tag identifies a number associated with the document, such as a form, serial, or order number.

Usage: The document number can occur only within the title page. This element is optional.

```
      .
      .
   :frontm
   :titlep
   :title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
   :docnum.MG-00001
   :date.July 3, 1978
   :author.L. T. Smith, Document&rbl.Administrator
   :address
   :aline.Any Company
   :aline.500 Main Street
   :aline.Anycity, Anyplace
   :eaddress
   :etitlep
   :preface
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF prints the contents of the document number element on the title page.

Results:

See "Appendix D: Title Page Example" on page 143 for an example.

DT -- DEFINITION TERM

Type	Attributes	Content	Termination
:dt	None	Text on Same Line (no other tags)	At End of Line

The **dt** tag identifies a word or phrase which is defined, described, or explained by an item of a definition list.

Usage: The definition term can occur only within a definition list (**dl**). It is repeatable. It must immediately be followed by its corresponding definition description element (**dd**).

```
:p.This paragraph precedes the list.
:dl
:dt.Term1
:dd.The first definition description.
This is an implied paragraph -- there was no :hp2.p:ehp2. tag.
:p.This paragraph is part of the first definition description.
:dt.Term2
:dd.The second definition description.
:lp.This list part introduces succeeding terms and descriptions.
:p.A list part could comment on preceding terms and descriptions.
:dt.Term3
:dd.The third definition description.
It has two topics:
:ol
:li.one topic;
:li.another topic.
:eol
:dt.Term4
:dd.The fourth definition description.
:edl
:p.This paragraph follows the list.
```

Processing With the "gdocprof" Profile: The supplied APF formats the definition terms and definition descriptions as a hanging indent list, with the indention equal to the specified term size (10M if the **tsize** attribute of the definition list is not specified). The terms are assumed to have a highlighting attribute of 2 if the **termhi** attribute of the definition list is not specified. (See "Highlighting" on page 9 for an explanation of highlighting.)

<u>Results:</u>

This paragraph precedes the list.

Term1 The first definition description. This is an implied paragraph -- there was no **p** tag.

This paragraph is part of the first definition description.

Term2 The second definition description.

This list part introduces succeeding terms and descriptions.

A list part could comment on preceding terms and descriptions.

Term3 The third definition description. It has two topics:

1. one topic;
2. another topic.

Term4 The fourth definition description.

This paragraph follows the list.

Type	Attributes	Content	Termination
:fig	id= depth= place= frame=	No Immediate Text	:efig

The **fig** tag identifies a diagram, table, or other illustration.

The **id** attribute label identifies the optional unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other figure can have the same unique identifier.

The **depth** attribute label identifies the optional depth attribute. Its value is an amount (in space units) of vertical white space to be included in the body of the figure.

The **place** attribute label identifies the optional placement attribute. Its value is the relationship of the figure to the surrounding text (either 'column' or 'inline' or 'page').

The **frame** attribute label identifies the optional frame attribute. Its value is the emphasis to be given the figure by framing it (either 'rule' or 'box' or 'none' or another character string).

Usage: The figure can occur wherever a basic document element can occur, except within the bounds of another figure, an example, a footnote, or a long quotation, or immediately after an element whose content is "Text on Same Line."

A figure contains a figure body (not explicitly tagged), and can also contain a figure caption (**figcap**) and a figure description (**figdesc**). When the figure is referred to by a figure reference element (**figref**), the **id** attribute and figure caption are mandatory. If the figure is to be listed automatically in the list of illustrations (**figlist**), the figure caption is mandatory.

The figure body can contain basic document elements (except figures) and "line elements." A line element is a line of a source document which is processed in such a way that it appears as an independent line in the output. Line elements can occur only within the bounds of a figure body or an example (**xmp**). There are no tags for a line element; one starts at the beginning of an input line and terminates at the end. A line element can contain text items.

Within the bounds of a figure body, elements that do not have explicit termination tags (**dd**, **li**, **lp**, **note**, **p**, and **pc**) are implicitly terminated at the end of the input line. Succeeding untagged input lines are treated as line elements. (As usual, the entered content may include **psc** elements to obtain special graphic effects, and .IM control words to imbed content from other files.)

If no content is entered, the APF will generate white space as specified by the **depth** attribute to provide room for artwork to be stripped in for reproduction.

The first part of the following example shows a figure whose body was generated by the APF. The second part is a figure whose body contains only line elements.

```
as seen in the following figure:
:fig place='inline' depth='1.5i' frame='box'
:figcap.Generated Vertical Space with Box Frame
:figdesc.Instead of entering the content of a figure body,
you can let the APF generate space for stripped-in art.
:efig
:pc.The figure below, in contrast, demonstrates that. . .
  .
  .
:fig
(1) A * B = C
(2) C = A * D
(3) B = D
:figcap.Line Elements
:efig
```

Processing With the "gdocprof" Profile: The supplied APF generates the required space if **depth** is specified. Otherwise, the content is processed without justification, running together of input lines, or spelling checking.

If placement is specified as "column" (a "column figure") or "inline," the figure is set at the same width as a column. If placement is specified as "page," the figure is set at the full width of the page. (If placement is not specified, "column" is assumed.)

Figures are "keeps"; that is, the output of processing a figure is intentionally kept together, and moved to the next column or page if there is insufficient room in the current column or page. Column figures "float"; if forced to the next column, text entered after the figure in the source document is brought ahead of it and set in the remainder of the current column. Inline figures do not float. Page figures float only during single-column formatting.

If **frame** is specified as "rule," horizontal rules are printed above and below the figure. If **frame** is specified as "box," rules are printed on all four sides. No rules are printed if **frame** is specified as "none." If **frame** is specified as any other character string, that string is printed repetitively as a border above and below the figure. (If not specified, "rule" is assumed.)

If **id** is specified and there is a figure caption, the next sequential figure number is generated (by the **figcap** APF) and is saved so **figref** elements can refer to it.

Note: A figure within the bounds of a list is not indented as are other basic document elements.

Results:

as seen in the following figure:

Figure 5. Generated Vertical Space with Box Frame: Instead of entering the content of a figure body, you can let the APF generate space for stripped-in art.

The figure below, in contrast, demonstrates that. . .

.

.

```
(1) A * B = C
(2) C = A * D
(3) B = D
```

Figure 6. Line Elements

FIGCAP -- FIGURE CAPTION

Type	Attributes	Content	Termination
:figcap	None	Text on Same Line (no other tags)	At End of Line

The **figcap** tag identifies the title or designation of a figure.

Usage: The figure caption can occur only in a figure element, after the figure body. If a reference is made to the figure by a figure reference element (**figref**), or if the figure is to be listed automatically in the list of illustrations (**figlist**), then the figure caption is mandatory. (However, the tag can be entered without text.) The figure caption text should be entered with initial capitals.

```
    .
    .
APFs associated with them, the profile, and the output device.
:fig id='se'.
    .
    .
:figcap.Markup Example
:figdesc.The source for the start of the
section :hdref refid='mupro'..
:efig.
    .
    .
```

Processing With the "gdocprof" Profile: The supplied APF generates the word "Figure" and the next sequential figure number, and prefixes them to the caption content (if any). The numbered caption is included in the list of illustrations (if any).

Results:

See Figure 4 on page 8 for an example.

FIGDESC -- FIGURE DESCRIPTION

Type	Attributes	Content	Termination
:figdesc	None	Implied P Structure	:efig

The **figdesc** tag identifies an extended comment or description of a figure.

Usage: The figure description can occur only in a figure, after the figure caption. It is optional. It contains an implied paragraph (no **p** tag), and may contain other basic document elements (except figures).

```
      .
      .
   :fig id='se'.
      .
      .
   :figcap.Markup Example
   :figdesc.The source for the start of the
   section :hdref.refid='mupro'..
   :efig.
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF formats the description, beginning on the same line as the figure caption.

Results:

See Figure 4 on page 8 for an example.

FIGLIST -- LIST OF ILLUSTRATIONS

Type	Attributes	Content	Termination
:figlist	None	Generated by APF	At End of Markup

The **figlist** tag identifies a listing of the figures in the document and the pages on which they occur.

Usage: The list of illustrations can occur only within the front matter, after the title page (and abstract, preface, and table of contents, if any). This element is optional.

```
   .
   .
:frontm
:titlep
   .
   .
:etitlep
:abstract
   .
   .
:preface
   .
   .
:toc
:figlist
:body
   .
   .
```

Processing With the "gdocprof" Profile: The supplied APF generates the content of the list of illustrations element from figure captions (**figcap**) in the document. It begins a new page (odd-numbered if the SYSVAR D YES option is specified). The TWOPASS option is required for normal processing of a General Document containing a **figlist**.

Note: The **figlist** may generate an incorrect page number for a figure which changes its output page during the second pass. This may be remedied by moving the figure to a point in the source document which appears on the same output page after both passes.

Results:

See this manual's list of illustrations for an example.

FIGREF -- FIGURE REFERENCE

Type	Attributes	Content	Termination
:figref	refid=	Generated by APF	At End of Markup

The **figref** tag identifies a reference to a figure, sometimes called a "figure callout."

The **refid** attribute label identifies the mandatory reference identifier attribute. Its value is the same as the **id** of the figure being referred to.

Usage: The figure reference can occur anywhere in text except in an element whose content is "Text on Same Line." It is a text item.

```
      .
      .
:p.:figref refid='se'. illustrates how starter set GML tags
were used in marking up parts of this book.
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF generates the content of the element, as shown in the example, below. The page number is included only if the figure is on a different page from the **figref**. The TWOPASS option is required when the reference precedes the figure.

Note: The **figref** may generate an incorrect page number if the figure it refers to changes its output page during the second pass. This may be remedied by moving the figure to a point in the source document which appears on the same output page after both passes.

Results:

Figure 4 on page 8 illustrates how starter set GML tags were used in marking up parts of this book.

FN -- FOOTNOTE

Type	Attributes	Content	Termination
:fn	id=	Implied P Structure	:efn

The **fn** tag identifies a note of reference, explanation, or comment, usually placed below the text on the printed page.

The **id** attribute label identifies the mandatory unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other footnote can have the same unique identifier.

Usage: The footnote can occur only in conjunction with a footnote reference (**fnref**). It may be entered wherever a basic document element could be entered, but not within the bounds of an example, a figure, a list, or another footnote, or immediately after an element whose content is "Text on Same Line." The footnote should be entered close to the **fnref** (so that both will appear on the same output page).

A footnote contains an implied paragraph (no **p** tag), and may also contain basic document elements (except figures and other footnotes). There must be no more than 10 lines of output within the bounds of a footnote.

```
:p.SCRIPT/VS saves text indicated as a footnote and
places it at the bottom of the
page.:fnref refid='emg'.
Subsequent footnotes are placed
below it.:fnref refid='adg'
:fn id='emg'.Up to 10 lines of output per footnote.
:efn
:fn id='adg'.Like this.
:efn
```

Processing With the "gdocprof" Profile: The supplied APF generates the next sequential footnote number, formats the footnote, keeps the processing output together, and places it below the text on the page. Footnotes are numbered in a single sequence throughout the document. The number is saved for reference by the related **fnref** element.

Results:

SCRIPT/VS saves text indicated as a footnote and places it at the bottom of the page. [7] Subsequent footnotes are placed below it.[8]

[7] Up to 10 lines of output per footnote.
[8] Like this.

FNREF -- FOOTNOTE REFERENCE

Type	Attributes	Content	Termination
:fnref	refid=	Generated by APF	At End of Markup

The **fnref** tag identifies a reference to a footnote.

The **refid** attribute label identifies the mandatory reference identifier attribute. Its value is the same as the **id** of the footnote being referred to.

Usage: The footnote reference can occur anywhere in text except in an element whose content is "Text on Same Line." It is a text item.

```
:p.SCRIPT/VS saves text indicated as a footnote and
places it at the bottom of the
page.:fnref refid='cfg'.
Subsequent footnotes are placed
below it.:fnref refid='ejm'
:fn id='cfg'.Up to 10 lines of output per footnote.
:efn
:fn id='ejm'.Like this.
:efn
```

Processing With the "gdocprof" Profile: The supplied APF generates the number of the footnote as the content of the **fnref** element. It appears in text as a superscript. The TWOPASS option is required when the reference precedes the footnote.

Results:

SCRIPT/VS saves text indicated as a footnote and places it at the bottom of the page. [9] Subsequent footnotes are placed below it.[10]

[9] Up to 10 lines of output per footnote.
[10] Like this.

FRONTM -- FRONT MATTER

Type	Attributes	Content	Termination
:frontm	None	No Immediate Text	:body

The **frontm** tag identifies that major element of a document which contains material that serves as a guide to the document's contents and nature, such as the title page, the abstract, and the table of contents.

Usage: The front matter can occur only at the start of the document, immediately after the **gdoc** tag. The front matter contains a title page, and may also contain an abstract, a preface, a table of contents, and a list of illustrations.

```
:gdoc sec='Company Confidential'
:frontm
:titlep
  .
  .
:body
  .
  .
:appendix
  .
  .
:backm
  .
  .
:egdoc
```

Processing With the "gdocprof" Profile: The supplied APF causes front matter pages to be numbered in Roman numerals. Front matter headings do not appear in the table of contents.

Results:

See the Roman numeral pages at the beginning of this manual for an example of the processing results.

GDOC -- GENERAL DOCUMENT

Type	Attributes	Content	Termination
:gdoc	sec=	No Immediate Text	:egdoc

The **gdoc** tag identifies a General Document.

The **sec** attribute label identifies the optional security level attribute. Its value is a character string that identifies the security classification of the document.

Usage: The General Document can occur only as a whole document; not as an element. The **gdoc** tag must be the first tag in the document.

```
    :gdoc sec='Company Confidential'
    :frontm
      .
      .
    :body
      .
      .
    :appendix
      .
      .
    :backm
      .
      .
    :egdoc
```

Processing With the "gdocprof" Profile: The supplied APF sets up the environment for the document. The security level, if specified, is printed as a running heading throughout the document, and on the title page. Any subsequent occurrences of **gdoc** (such as within imbedded files) are ignored.

Results:

This manual is an example of a General Document (except for the tag descriptions and some other elements that occur in GML Markup Guides).

HDREF -- HEADING REFERENCE

Type	Attributes	Content	Termination
:hdref	refid=	Generated by APF	At End of Markup

The **hdref** tag identifies a reference to a heading.

The **refid** attribute label identifies the mandatory reference identifier attribute. Its value is the same as the **id** of the heading being referred to.

Usage: The heading reference can occur anywhere in text except in an element whose content is "Text on Same Line." It is a text item.

```
   .
   .
If there is insufficient space, you can use the technique
for attribute insertion
described in the section on :hdref refid='ral'..
   .
   .
```

Processing With the "gdocprof" Profile: The supplied APF generates the content of the element. The content consists of the text of the heading referred to, plus the page number if the heading is on a different page from the **hdref**. The TWOPASS option is required when the reference precedes the heading.

Results:

If there is insufficient space, you can use the technique for attribute insertion described in the section on "Inserting Attributes (.ATT)" on page 14.

HP0 -- HIGHLIGHTED PHRASE (TYPE 0)

Type	Attributes	Content	Termination
:hp0	None	Text Item	:ehp0

The **hp0** tag identifies a phrase whose highlight value is "0." A phrase is technically a series of words expressing a thought in a fragmentary manner. In practice, though, a highlighted phrase can be one or more related words which are to be distinguished from their surrounding text for some reason. (See "Highlighting" on page 9 for an explanation of highlighting.)

Usage: The highlighted phrase (type 0) can occur anywhere in text except in an element whose content is "Text on Same Line."

Note: The starter set includes four "highlighted phrase" element types (**hp0**, **hp1**, **hp2**, and **hp3**) which simplify the markup of phrases (**ph**) with a **hi** attribute.

```
:p.In the starter set, Highlight 0 is set like normal text.
:hp2.It can be used for a :hp0.deemphasized
element:ehp0. that occurs within an
emphasized element:ehp2..
```

Processing With the "gdocprof" Profile: The supplied APF prints the phrase in a font and style appropriate to its highlighting. Type zero highlighting is printed in the same font and style as normal body text. It is used for contrast when a deemphasized phrase is nested within an emphasized phrase. The text surrounding the phrase is unaffected.

Results:

In the starter set, Highlight 0 is set like normal text. **It can be used for a** deemphasized element **that occurs within an emphasized element.**

HP1 -- HIGHLIGHTED PHRASE (TYPE 1)

Type	Attributes	Content	Termination
:hp1	None	Text Item	:ehp1

The **hp1** tag identifies a phrase whose highlight value is "1." A phrase is technically a series of words expressing a thought in a fragmentary manner. In practice, though, a highlighted phrase can be one or more related words which are to be distinguished from their surrounding text for some reason. (See "Highlighting" on page 9 for an explanation of highlighting.)

Usage: The highlighted phrase (type 1) can occur anywhere in text except in an element whose content is "Text on Same Line."

Note: The starter set includes four "highlighted phrase" element types (**hp0**, **hp1**, **hp2**, and **hp3**) which simplify the markup of phrases (**ph**) with a **hi** attribute.

```
:p.Four kinds of highlighting are possible with the starter set.
They are:
:hp0.Highlight 0:ehp0.,
:hp1.Highlight 1:ehp1.,
:hp2.Highlight 2:ehp2., and
:hp3.Highlight 3:ehp3..
```

Processing With the "gdocprof" Profile: The supplied APF prints the phrase in a font and style appropriate to its highlighting. The text surrounding the phrase is unaffected.

Results:

Four kinds of highlighting are possible with the starter set. They are: Highlight 0, Highlight 1, **Highlight 2**, and **Highlight 3**.

HP2 -- HIGHLIGHTED PHRASE (TYPE 2)

Type	Attributes	Content	Termination
:hp2	None	Text Item	:ehp2

The **hp2** tag identifies a phrase whose highlight value is "2." A phrase is technically a series of words expressing a thought in a fragmentary manner. In practice, though, a highlighted phrase can be one or more related words which are to be distinguished from their surrounding text for some reason. (See "Highlighting" on page 9 for an explanation of highlighting.)

Usage: The highlighted phrase (type 2) can occur anywhere in text except in an element whose content is "Text on Same Line."

Note: The starter set includes four "highlighted phrase" element types (**hp0**, **hp1**, **hp2**, and **hp3**) which simplify the markup of phrases (**ph**) with a **hi** attribute.

```
:p.Four kinds of highlighting are possible with the starter set.
They are:
:hp0.Highlight 0:ehp0.,
:hp1.Highlight 1:ehp1.,
:hp2.Highlight 2:ehp2., and
:hp3.Highlight 3:ehp3..
```

Processing With the "gdocprof" Profile: The supplied APF prints the phrase in a font and style appropriate to its highlighting. The text surrounding the phrase is unaffected.

Results:

Four kinds of highlighting are possible with the starter set. They are: Highlight 0, Highlight 1, **Highlight 2**, and **Highlight 3**.

HP3 -- HIGHLIGHTED PHRASE (TYPE 3)

Type	Attributes	Content	Termination
:hp3	None	Text Item	:ehp3

The **hp3** tag identifies a phrase whose highlight value is "3." A phrase is technically a series of words expressing a thought in a fragmentary manner. In practice, though, a highlighted phrase can be one or more related words which are to be distinguished from their surrounding text for some reason. (See "Highlighting" on page 9 for an explanation of highlighting.)

Usage: The highlighted phrase (type 3) can occur anywhere in text except in an element whose content is "Text on Same Line."

Note: The starter set includes four "highlighted phrase" element types (**hp0**, **hp1**, **hp2**, and **hp3**) which simplify the markup of phrases (**ph**) with a **hi** attribute.

```
:p.Four kinds of highlighting are possible with the starter set.
They are:
:hp0.Highlight 0:ehp0.,
:hp1.Highlight 1:ehp1.,
:hp2.Highlight 2:ehp2., and
:hp3.Highlight 3:ehp3..
```

Processing With the "gdocprof" Profile: The supplied APF prints the phrase in a font and style appropriate to its highlighting. The text surrounding the phrase is unaffected.

Results:

Four kinds of highlighting are possible with the starter set. They are: Highlight 0, Highlight 1, **Highlight 2**, and **Highlight 3**.

H0 -- HEAD ZERO

Type	Attributes	Content	Termination
:h0	id= stitle=	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h0** tag identifies a zero-level heading. The heading begins a zero-level heading segment, also known as a "part." A part is a separately identified group of consecutive chapters.

The **id** attribute label identifies the optional unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other heading (of any type) can have the same unique identifier.

The **stitle** attribute label identifies the optional short title attribute. Its value is a character string that is a short version of the heading.

Usage: The head zero can occur only within the body of a document. The heading text should be entered with initial capitals. When the heading is referred to by a heading reference element (**hdref**), the **id** attribute is mandatory.

A part contains one or more chapters. It also may contain basic document elements preceding the first chapter. The part segment is terminated by another part segment, or by starting the appendix section or back matter.

```
    .
    .
.att stitle='GML Markup Guide'
:h0 id='c3'.Part One: GML Markup Guide for General Documents
:h1.What is GML markup?
    .
    .
```

Processing With the "gdocprof" Profile: The supplied APF begins a new page (odd-numbered if the SYSVAR D YES option is specified). The heading is aligned at the right margin. The heading text is included in the table of contents (if any). If the **id** attribute is specified, the text of the heading and the page number on which it appears are saved for reference by a **hdref** element. The short title (or the heading text, if no short title is specified) is printed as a running footing on odd-numbered pages throughout the part until replaced by the heading (or short title) of the first chapter.

<u>Results:</u>

<u>The heading "Part One: GML Markup Guide for General Documents" on page 1 is an example.</u>

H1 -- HEAD ONE

Type	Attributes	Content	Termination
:h1	id= stitle=	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h1** tag identifies a first-level heading which begins a first-level heading segment. Within the body of a document, a first-level heading segment is also called a "chapter."

The **id** attribute label identifies the optional unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other heading (of any type) can have the same unique identifier.

The **stitle** attribute label identifies the optional short title attribute. Its value is a character string that is a short version of the heading.

Usage: The head one can occur within the appendix, back matter, or a part, or directly within the body if the body is not divided into parts. The text of the heading should be entered with initial capitals. When the heading is referred to by a heading reference element (**hdref**), the **id** attribute is mandatory.

The heading segment can contain basic document elements and lower-level heading segments. It is terminated by another first-level heading segment (**h1**), a part (**h0**), or the appendix or back matter.

```
  .
  .
:h1 id='mupro'.Markup Procedures
:p.This section explains:
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF begins a new page (odd-numbered if the SYSVAR D YES option is specified). The heading text is included in the table of contents (if any). If the **id** attribute is specified, the text of the heading and the page number on which it appears are saved for reference by a **hdref** element. The short title (or the heading text, if no short title is specified) is printed as a running footing on odd-numbered pages throughout the chapter. The heading is numbered automatically if the SYSVAR H NUM option is specified.

<u>Results:</u>

<u>The heading "Markup Procedures" on page 11 is an example.</u>

Type	Attributes	Content	Termination
:h2	id=	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h2** tag identifies a second-level heading which begins a second-level heading segment. A document is divided into segments to guide the reader when the material so requires, as when first-level heading segments are long and the subject complex.

The **id** attribute label identifies the optional unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other heading (of any type) can have the same unique identifier.

Usage: The head two can occur within an abstract, preface, or higher-level (except 0) heading segment. The text of the heading should be entered with initial capitals. When the heading is referred to by a heading reference element (**hdref**), the **id** attribute is mandatory.

The heading segment can contain basic document elements and lower-level heading segments. It is terminated by another heading segment of the same level or higher, by a front matter segment, or by the body, appendix or back matter.

```
    .
    .
symbols you may find useful in your documents.
:h2 id='mucw3'.Control Words and Macros
:p.There are a number of situations in which it will be . . .
    .
    .
```

Processing With the "gdocprof" Profile: The supplied APF includes the heading in the table of contents (if any). If the **id** attribute is specified, the text of the heading and the page number on which it appears are saved for reference by a **hdref** element. The heading is numbered automatically if the SYSVAR H NUM option is specified.

Results:

The heading "Control Words and Macros" on page 13 is an example.

H3 -- HEAD THREE

Type	Attributes	Content	Termination
:h3	id=	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h3** tag identifies a third-level heading which begins a third-level heading segment. A document is divided into segments to guide the reader when the material so requires, as when chapters are long and the subject complex.

The **id** attribute label identifies the optional unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other heading (of any type) can have the same unique identifier.

Usage: The head three can occur within an abstract, preface, or higher-level (except 0) heading segment. The text of the heading should be entered with initial capitals. When the heading is referred to by a heading reference element (**hdref**), the **id** attribute is mandatory.

The heading segment can contain basic document elements and lower-level heading segments. It is terminated by another heading segment of the same level or higher, by a front matter segment, or by the body, appendix or back matter.

```
      .
      .
direct entry of control words can provide.
:h3 id='sdm'.Source Document Management
:p.You may want to store parts of your document as . . .
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF includes the heading in the table of contents (if any). If the **id** attribute is specified, the text of the heading and the page number on which it appears are saved for reference by a **hdref** element. The heading is numbered automatically if the SYSVAR H NUM option is specified.

Results:

The heading "Source Document Management" on page 13 is an example.

Type	Attributes	Content	Termination
:h4	id=	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h4** tag identifies a fourth-level heading which begins a fourth-level heading segment. A document is divided into segments to guide the reader when the material so requires, as when chapters are long and the subject complex.

The **id** attribute label identifies the optional unique identifier attribute. Its value is a string of up to five alphameric characters which can be used to refer to this element. No other heading (of any type) can have the same unique identifier.

Usage: The head four can occur within an abstract, preface, or higher-level (except 0) heading segment. The text of the heading should be entered with initial capitals. When the heading is referred to by a heading reference element (**hdref**), the **id** attribute is mandatory.

The heading segment can contain basic document elements and lower-level heading segments. It is terminated by another heading segment of the same level or higher, by a front matter segment, or by the body, appendix or back matter.

```
      .
      .
for doing this.
:h4 id='ral'.Inserting Attributes (.att)
:p.Sometimes you may not have enough room on one line
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF includes the heading in the table of contents (if any). If the **id** attribute is specified, the text of the heading and the page number on which it appears are saved for reference by a **hdref** element. The heading is numbered automatically if the SYSVAR H NUM option is specified.

Results:

The heading "Inserting Attributes (.ATT)" on page 14 is an example.

H5 -- HEAD FIVE

Type	Attributes	Content	Termination
:h5	None	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h5** tag identifies a fifth-level heading which begins a fifth-level heading segment. A document is divided into segments to guide the reader when the material so requires, as when chapters are long and the subject complex.

Usage: The head five can occur within an abstract, preface, or higher-level (except 0) heading segment. The text of the heading should be entered with initial capitals.

The heading segment can contain basic document elements and lower-level heading segments. It is terminated by another heading segment of the same level or higher, by a front matter segment, or by the body, appendix or back matter.

```
    .
    .
end of the previous paragraph unit.
:h5.Head Five Example
:p.It is run into the following paragraph.
    .
    .
:h5.Another Head Five
:note.It stands alone because it is not followed by a paragraph.
    .
    .
```

Processing With the "gdocprof" Profile: The supplied APF formats the heading as a run-in head if the next element is a paragraph (**p**). Otherwise, the heading is set as an independent line.

Results:

end of the previous paragraph unit.

HEAD FIVE EXAMPLE: It is run into the following paragraph.

ANOTHER HEAD FIVE

Note: It stands alone because it is not followed by a paragraph.

H6 -- HEAD SIX

Type	Attributes	Content	Termination
:h6	None	Text on Same Line (no other tags)	At End of Line See "Usage" in text

The **h6** tag identifies a sixth-level heading which begins a sixth-level heading segment. A document is divided into segments to guide the reader when the material so requires, as when chapters are long and the subject complex.

Usage: The head six can occur within an abstract, preface, or higher-level (except 0) heading segment. The text of the heading should be entered with initial capitals.

The heading segment can contain basic document elements. It is terminated by another heading segment of the same level or higher, by a front matter segment, or by the body, appendix or back matter.

```
      .
      .
end of the previous paragraph unit.
:h6.Head Six Example
:p.It is run into the following paragraph.
      .
      .
:h6.Another Head Six
:note.It stands alone because it is not followed by a paragraph.
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF formats the heading as a run-in head if the next element is a paragraph (p). Otherwise, the heading is set as an independent line.

<u>Results:</u>

end of the previous paragraph unit.

<u>Head Six Example</u>: It is run into the following paragraph.

<u>Another Head Six</u>

<u>Note</u>: It stands alone because it is not followed by a paragraph.

LI -- LIST ITEM

Type	Attributes	Content	Termination
:li	None	Implied P Structure	:li, :lp, :eol, :esl, or :eul

The **li** tag identifies an element of an ordered list, simple list, or unordered list.

Usage: The list item can occur only within an ordered list, simple list, or unordered list. It is repeatable. It contains an implied paragraph (no **p** tag), and may also contain basic document elements.

List items of an ordered list should be entered without sequence numbers or letters, as these will be provided by the APF.

List items of an unordered list should be entered without bullets or dashes, as these will be provided by the APF.

```
   .
   .
:p.This paragraph precedes the list.
:ol
:li.The first list item.
:p.This paragraph is part of the first list item.
:li.The second list item.
:lp.This list part comments on the preceding list items.
:p.A list part can also
introduce the succeeding list items.
:li.The third list item.
It has two topics:
:ul
:li.one topic;
:li.another topic.
:eul
:li.The fourth list item.
:eol
:p.This paragraph follows the list.
   .
   .
```

Processing With the "gdocprof" Profile: The supplied APF prefixes the list item with a number or bullet (if required by the type of list) and formats it.

Results:

This paragraph precedes the list.

1. The first list item.

   This paragraph is part of the first list item.

2. The second list item.

This list part comments on the preceding list items.

A list part can also introduce the succeeding list items.

3. The third list item. It has two topics:

   - one topic;

   - another topic.

4. The fourth list item.

This paragraph follows the list.

See "Lists" on page 8 for additional examples of list items.

LP -- LIST PART

Type	Attributes	Content	Termination
:lp	None	Implied P Structure	:dt, :li, :edl, :eol, :esl, or :eul

The **lp** tag identifies a comment or explanation which applies to part of a list (that is, to some of the list's items).

Usage: The list part can occur only within a list. It is optional and repeatable. It contains an implied paragraph (no **p** tag), and may also contain basic document elements.

```
      .
      .
    :p.This paragraph precedes the list.
    :ol
    :li.The first list item.
    :p.This paragraph is part of the first list item.
    :li.The second list item.
    :lp.This list part comments on the preceding list items.
    :p.A list part can also
    introduce the succeeding list items.
    :li.The third list item.
    It has two topics:
    :ul
    :li.one topic;
    :li.another topic.
    :eul
    :li.The fourth list item.
    :eol
    :p.This paragraph follows the list.
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF aligns the list part with the number or bullet of the list items (or with the definition term) and formats it.

<u>Results:</u>

This paragraph precedes the list.

1. The first list item.

   This paragraph is part of the first list item.

2. The second list item.

This list part comments on the preceding list items.

A list part can also introduce the succeeding list items.

3. The third list item. It has two topics:

   - one topic;

   - another topic.

4. The fourth list item.

This paragraph follows the list.

LQ -- LONG QUOTATION

Type	Attributes	Content	Termination
:lq	None	No Immediate Text	:elq

The **lq** tag identifies an excerpt, or "block quotation," which is set off from other text to show that it is quoted from another source.

Usage: The long quotation can occur wherever a basic document element can occur. It contains basic document elements (except **fig**).

```
    :p.The following is an excerpt from the markup rules
    for long quotations.
    Observe that the long quotation is
    indented on both the left and the right.
    :lq
    :p.Do not enclose long quotations in quotation marks.
    Short quotations within a long quotation should be
    marked with the :hp2.q:ehp2. and :hp2.eq:ehp2. tags.
    :p.Long quotations can include lists and other basic document elements
    (except figures).
    :elq
    :p.This paragraph follows the long quotation.
```

Processing With the "gdocprof" Profile: The supplied APF indents the long quotation.

Results:

The following is an excerpt from the markup rules for long quotations. Observe that the long quotation is indented on both the left and the right.

> Do not enclose long quotations in quotation marks. Short quotations within a long quotation should be marked with the **q** and **eq** tags.
>
> Long quotations can include lists and other basic document elements (except figures).

This paragraph follows the long quotation.

NOTE -- NOTE

Type	Attributes	Content	Termination
:note	None	Paragraph Unit	Paragraph or Higher Level Element

The **note** tag identifies a paragraph containing a comment or explanation which must be called to the reader's attention.

Usage: The note can occur wherever a basic document element can occur.

```
    .
    .
:note.This is the first example of a note element.
It is normally printed flush with the left margin.
:ol
:li.This is an ordered list item.
:note.This note is part of a list item in an ordered list.
Notice the indention level in relation to the text of the
list item.
:eol
:note.This is the first note after ending the ordered list.
    .
    .
```

Processing With the "gdocprof" Profile: The supplied APF causes the word "Note:" to be prefixed to a paragraph which is formatted flush left. Notes within elements that are indented, such as list items and examples, are formatted at the appropriate indention of the outer element. Notes are set in block style.

Results:

Note: This is the first example of a note element. It is normally printed flush with the left margin.

1. This is an ordered list item.

   Note: This note is part of a list item in an ordered list. Notice the indention level in relation to the text of the list item.

Note: This is the first note after ending the ordered list.

OL -- ORDERED LIST

Type	Attributes	Content	Termination
:ol	None	No Immediate Text	:eol

The **ol** tag identifies a list of elements, called "list items," whose order is significant. It is usually printed with sequence numbers or letters to emphasize the order of the items.

Usage: The ordered list can occur wherever a basic document element can occur. It contains a series of list items (**li**), which may be intermixed with individual elements known as "list parts" (**lp**).

All types of list (definition, ordered, unordered, and simple) can occur within the bounds of one another), but within the bounds of a single list the level of nesting cannot be more than 3 for unordered lists, 6 for definition lists, 6 for simple lists, and 6 for ordered lists.

List items of an ordered list should be entered without sequence numbers or letters, as these will be provided by the APF.

```
  .
  .
:p.This paragraph precedes the list.
:ol
:li.The first list item.
:p.This paragraph is part of the first list item.
:li.The second list item.
:lp.This list part comments on the preceding list items.
:p.A list part can also
introduce the succeeding list items.
:li.The third list item.
It has two topics:
:ul
:li.one topic;
:li.another topic.
:eul
:li.The fourth list item.
:eol
:p.This paragraph follows the list.
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF indents the list appropriately to its level of nesting, and resets the item counter for the first list item. List items will be numbered or lettered sequentially in different styles, according to the level of nesting.

Results:

This paragraph precedes the list.

1. The first list item.

   This paragraph is part of the first list item.

2. The second list item.

This list part comments on the preceding list items.

A list part can also introduce the succeeding list items.

3. The third list item. It has two topics:

   - one topic;

   - another topic.

4. The fourth list item.

This paragraph follows the list.

See "Lists" on page 8 for additional examples of lists.

P -- PARAGRAPH

Type	Attributes	Content	Termination
:p	None	Paragraph Unit	Paragraph or Higher Level Element

The **p** tag identifies a paragraph -- that is, one or more sentences related by their subject matter.

Usage: The paragraph can occur wherever a basic document element can occur.

```
      .
      .
   :p.This is the first example of a paragraph element.
   It is normally printed flush with the left margin.
   :ol
   :li.This is an ordered list item.
   :p.This paragraph is part of a list item in an ordered list.
   Notice the indention level in relation to the text of the
   list item.
   :eol
   :p.This is the first paragraph after ending the ordered list.
      .
      .
```

Processing With the "gdocprof" Profile: The supplied APF causes paragraphs to be formatted flush left. Paragraphs within elements that are indented, such as list items and examples, are formatted at the appropriate indention of the outer element. Explicitly tagged paragraphs are set in block style. The format of the implied paragraph of a **dd**, **figdesc**, **fn**, **li** or **lp** element is determined by the format of that element.

Results:

This is the first example of a paragraph element. It is normally printed flush with the left margin.

1. This is an ordered list item.

   This paragraph is part of a list item in an ordered list. Notice the indention level in relation to the text of the list item.

This is the first paragraph after ending the ordered list.

PC -- PARAGRAPH CONTINUATION

Type	Attributes	Content	Termination
:pc	None	Paragraph Unit	Paragraph or Higher Level Element

The **pc** tag identifies a paragraph continuation -- that is, one or more sentences related by their subject matter to a paragraph which has been interrupted by an address, example, figure, list, or long quotation.

Usage: The paragraph continuation can occur after the sequence consisting of a paragraph unit followed by an address, example, figure, list, or long quotation.

```
  .
  .
:p.The subject of a paragraph might be continued through
:sl
:li.an address, a list,
:li.an example or figure, or
:li.a long quotation,
:esl
:pc.and continue to be discussed in flowing text.
The discussion could continue indefinitely through
:ol
:li.other addresses or lists,
:li.other examples or figures, and
:li.other long quotations,
:eol
:pc.by adding additional paragraph continuations, like this.
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF causes paragraph continuations to be formatted flush left. Paragraph continuations within elements that are indented, such as list items, examples, and figures, are formatted at the appropriate indention of the outer element. Paragraph continuations are set in block style (even where the paragraph units being continued have their first line indented).

Results:

The subject of a paragraph might be continued through

    an address, a list,

    an example or figure, or

    a long quotation,

and continue to be discussed in flowing text. The discussion could continue indefinitely through

1. other addresses or lists,

2. other examples or figures, and

3. other long quotations,

by adding additional paragraph continuations, like this.

Type	Attributes	Content	Termination
:ph	hi=	Text Item	:eph

The **ph** tag identifies a phrase -- technically a series of words expressing a thought in a fragmentary manner. In practice, a phrase can be one or more related words which are to be distinguished from their surrounding text for some reason.

The **hi** attribute label identifies the optional highlight (emphasis) attribute. Its value is a number from 0 to 3 which identifies the type of emphasis (if any) associated with the phrase. (See "Highlighting" on page 9 for an explanation of highlighting.)

Usage: The phrase can occur anywhere in text except within the bounds of another phrase (**ph**), or in an element whose content is "Text on Same Line."

Note: The starter set includes four "highlighted phrase" element types (**hp0**, **hp1**, **hp2**, and **hp3**) which simplify the markup of phrases (**ph**) with a **hi** attribute.

```
:p.Four kinds of highlighting are possible with the starter set.
They are:
:ph hi='0'.Highlight 0:eph.,
:ph hi='1'.Highlight 1:eph.,
:ph hi='2'.Highlight 2:eph., and
:ph hi='3'.Highlight 3:eph..
```

Processing With the "gdocprof" Profile: The supplied APF prints the phrase in the normal body text font and style unless highlighting is specified. The text surrounding the phrase is unaffected.

Note: Your document administrator may have defined other attributes which can be associated with phrases.

Results:

Four kinds of highlighting are possible with the starter set. They are: Highlight 0, Highlight 1, **Highlight 2**, and **Highlight 3**.

PREFACE -- PREFACE

Type	Attributes	Content	Termination
:preface	None	No Immediate Text	:toc, :figlist, or :body

The **preface** tag identifies the writer's introductory remarks, such as acknowledgements, or reasons for writing the document.

Usage: The preface can occur only within the front matter, after the title page (and abstract, if any). It can contain basic document elements and second-level and lower heading segments. This element is optional.

```
  .
  .
:frontm
:titlep
  .
  .
:etitlep
:abstract
  .
  .
:preface
:p.There has long been a need for a method of bringing
typographic capabilities to the production of office documents,
using normal secretarial skills.
  .
  .
:toc
:figlist
:body
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF begins a new page (odd-numbered if the SYSVAR D YES option is specified) and generates the word "Preface" as a heading.

Results:

See the preface of this manual for an example.

PSC -- PROCESS-SPECIFIC CONTROLS

Type	Attributes	Content	Termination
:psc	proc=	See "Usage" in text	:epsc

The **psc** tag identifies an element which contains control words and text which are meaningful for one or more specific processes that will be performed on the document.

The **proc** attribute label identifies the optional process attribute. Its value is an uppercase alphameric character string which identifies one or more processes to which the element is restricted. Process names may be 1 to 8 characters long, and must be separated by blanks.

Usage: The process-specific controls can occur potentially anywhere in a document (even within a word), but this depends upon the material you enter into it. It may contain any control words or text that can be passed through the SCRIPT/VS system, including markup for "post-processors" that will read the output of the SCRIPT/VS processing.

See "Graphic Formatting" on page 14, "Modifying Processing Results" on page 14, and "Additional General Processing" on page 15 for information about the use of **psc** elements.

The following example shows how a figure which is too large for a single page when printed at 6 lines per inch can conditionally be split into two figures. The markup shown will cause Figure 18 on page 133 to print as two separate figures when printing is done at 6 lines per inch, and as a single figure when printing is done at 8 or 12 lines per inch.

```
:fig id='gdoc' place='inline'
:psc
.cm graphic
.se @ssk = 0
:epsc
:ul
:li.Overall Structure:
(additional text)
:psc proc='1403N6 1403W6 1403SW 3800N6 3800W6 3800N6S 3800W6S 3800N8S 3800W8S TERM'
.cm graphic
.cm split figure for 6 line/inch devices
:eul
:figcap.Summary of General Document Tags (Part 1 of 2)
:efig
:fig
:ul
:epsc
:li.Basic Document Elements:
(additional text)
:eul
:psc
.cm graphic
.se @ssk = 1
:epsc
:psc proc='1403N6 1403W6 1403SW 3800N6 3800W6 3800N6S 3800W6S 3800N8S 3800W8S TERM'
.cm graphic
.cm split figure for 6 line/inch devices
:figcap.Summary of General Document Tags (Part 2 of 2)
:epsc
:psc proc='1403N8 1403W8 3800N8 3800W8 3800W12 3800W12S'
.cm graphic
.cm caption for "unsplit" 8 or 12 lines per inch version of figure
:figcap.Summary of General Document Tags
:epsc
:efig
```

Processing With the "gdocprof" Profile: The supplied APF will cause SCRIPT/VS to ignore a **psc** element with a specified **proc** value which does not include the process name or names associated with the SCRIPT/VS run. A process name can be designated with the SYSVAR P option. If this option is not specified, the logical device specified in the SCRIPT command, and its underlying physical device, are both treated as default process names. Figure 19 on page 140 contains the starter set logical and physical device names. (Your document administrator may designate other process names, or change the default.) If **proc** is not specified, the **psc** is valid for all processes.

Results:

See Figure 18 on page 133 for the results.

Type	Attributes	Content	Termination
:q	None	Text Item	:eq

The **q** tag identifies a phrase in which the exact words of a person or text are cited.

Usage: The quote can occur anywhere in text except in an element whose content is "Text on Same Line." Trailing punctuation which is not really part of the quote should not be included in the element. It should be entered after the **eq** tag, as in the example.

```
:p.Here is :q.an example of a
quoted phrase with :q.another quoted
phrase:eq. nested within it:eq..
Note the two periods at the end: one ends the :q.markup:eq.,
while the other ends the :q.sentence:eq..
```

Processing With the "gdocprof" Profile: The supplied APF prints the phrase within double quotation marks. Nested quoted phrases will alternate single quotes with double quotes.

A comma or period immediately following the **eq** tag will be brought inside the closing quotation mark. (Your document administrator may have changed the profile to prevent this if your installation follows a different style.)

<u>Results:</u>

Here is "an example of a quoted phrase with 'another quoted phrase' nested within it." Note the two periods at the end: one ends the "markup," while the other ends the "sentence."

SL -- SIMPLE LIST

Type	Attributes	Content	Termination
:sl	None	No Immediate Text	:esl

The **sl** tag identifies a list of elements, called "list items." It differs from an unordered list in that the items themselves are usually short and not emphasized, and so are usually not printed with bullets or dashes to distinguish them.

Usage: The simple list can occur wherever a basic document element can occur. It contains a series of list items (**li**), which may be intermixed with individual elements known as "list parts" (**lp**).

All types of list (definition, ordered, unordered, and simple) can occur within the bounds of one another), but within the bounds of a single list the level of nesting cannot be more than 3 for unordered lists, 6 for definition lists, 6 for simple lists, and 6 for ordered lists.

```
  .
  .
:p.This paragraph precedes the list.
:sl
:li.The first list item.
:p.This paragraph is part of the first list item.
:li.The second list item.
:lp.This list part comments on the preceding list items.
:p.A list part can also
introduce the succeeding list items.
:li.The third list item.
It has two topics:
:sl
:li.one topic;
:li.another topic.
:esl
:li.The fourth list item.
:esl
:p.This paragraph follows the list.
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF indents the list appropriately to its level of nesting. List items will be "bulleted" in different styles, according to the level of nesting.

Results:

This paragraph precedes the list.

    The first list item.

    This paragraph is part of the first list item.

    The second list item.

This list part comments on the preceding list items.

A list part can also introduce the succeeding list items.

    The third list item. It has two topics:

        one topic;

        another topic.

    The fourth list item.

This paragraph follows the list.

See "Lists" on page 8 for additional examples of lists.

TITLE -- DOCUMENT TITLE

Type	Attributes	Content	Termination
:title	stitle=	Text on Same Line (no other tags)	At End of Line

The **title** tag identifies the name of the document.

The **stitle** attribute label identifies the optional short title attribute. Its value is a character string that is a short version of the document title.

Usage: The document title can occur only as the first element of the title page. The text of the document title should be entered with initial capitals.

```
   .
   .
 :frontm
 :titlep
 :title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
 :docnum.MG-00001
 :date.July 3, 1978
 :author.L. T. Smith, Document&rbl.Administrator
 :address
 :aline.Any Company
 :aline.500 Main Street
 :aline.Anycity, Anyplace
 :eaddress
 :etitlep
 :preface
   .
   .
```

Processing With the "gdocprof" Profile: The supplied APF prints the document title on the title page. The short title (or the full document title, if no short title is specified) is printed as a running footing on even-numbered pages throughout the document.

Results:

See "Appendix D: Title Page Example" on page 143 for an example.

TITLEP -- TITLE PAGE

Type	Attributes	Content	Termination
:titlep	None	No Immediate Text	:etitlep

The **titlep** tag identifies the element of the front matter which contains general information about the document that is to appear on the title page.

Usage: The title page can occur only as the first element of the front matter. It contains the document title and document date, and can contain a document number, author, and address.

```
       .
       .
:frontm
:titlep
:title stitle='GML User''s Guide'.GML Markup and Processing&rbl.Guide
:docnum.MG-00001
:date.July 3, 1978
:author.L. T. Smith, Document&rbl.Administrator
:address
:aline.Any Company
:aline.500 Main Street
:aline.Anycity, Anyplace
:eaddress
:etitlep
:preface
       .
       .
```

Processing With the "gdocprof" Profile: The supplied APF prints the contents of the title page (unless title page printing is suppressed by use of the SYSVAR T option). If the document has no date element, the APF will print a system-supplied date of processing on the title page.

Results:

See "Appendix D: Title Page Example" on page 143 for an example.

TOC -- TABLE OF CONTENTS

Type	Attributes	Content	Termination
:toc	None	Generated by APF	At End of Markup

The **toc** tag identifies a listing of the sections of the document and the pages on which they begin.

Usage: The table of contents can occur only within the front matter, after the title page (and abstract and preface, if any). This element is optional.

```
    .
    .
:frontm
:titlep
    .
    .
:etitlep
:abstract
    .
    .
:preface
    .
    .
:toc
:figlist
:body
    .
    .
```

Processing With the "gdocprof" Profile: The supplied APF begins a new page (odd-numbered if SYSVAR D YES is specified). It generates the content of the table of contents element from the text of headings (h0 through h4) in the body, appendix section (if any), and back matter (if any). The TWOPASS option is required for normal processing of a General Document containing a **toc**.

Results:

See the table of contents of this manual for an example.

UL -- UNORDERED LIST

Type	Attributes	Content	Termination
:ul	None	No Immediate Text	:eul

The **ul** tag identifies a list of elements, called "list items," whose order is not significant. The items themselves are usually lengthy or emphasized, and are therefore usually printed with preceding bullets or dashes to distinguish them.

Usage: The unordered list can occur wherever a basic document element can occur. It contains a series of list items (**li**), which may be intermixed with individual elements known as "list parts" (**lp**).

All types of list (definition, ordered, unordered, and simple) can occur within the bounds of one another), but within the bounds of a single list the level of nesting cannot be more than 3 for unordered lists, 6 for definition lists, 6 for simple lists, and 6 for ordered lists.

List items of an unordered list should be entered without bullets or dashes, as these will be provided by the APF.

```
   .
   .
:p.This paragraph precedes the list.
:ul
:li.The first list item.
:p.This paragraph is part of the first list item.
:li.The second list item.
:lp.This list part comments on the preceding list items.
:p.A list part can also
introduce the succeeding list items.
:li.The third list item.
It has two topics:
:ol
:li.one topic;
:li.another topic.
:eol
:li.The fourth list item.
:eul
:p.This paragraph follows the list.
   .
   .
```

Processing With the "gdocprof" Profile: The supplied APF indents the list appropriately to its level of nesting. List items will be "bulleted" in different styles, according to the level of nesting.

Results:

This paragraph precedes the list.

- The first list item.

  This paragraph is part of the first list item.

- The second list item.

This list part comments on the preceding list items.

A list part can also introduce the succeeding list items.

- The third list item. It has two topics:

  1. one topic;

  2. another topic.

- The fourth list item.

This paragraph follows the list.

See "Lists" on page 8 for additional examples of lists.

Type	Attributes	Content	Termination
:xmp	depth=	No Immediate Text	:exmp

The **xmp** tag identifies an example, such as a diagram, table, or other illustration, which is read in line with the main text. It differs from a figure in that it is not captioned, or referred to from other parts of the document.

The **depth** attribute label identifies the optional depth attribute. Its value is an amount (in space units) of vertical white space to be included in the example.

Usage: The example can occur wherever a basic document element can occur.

The example can contain basic document elements (except figures and other examples) and "line elements." A line element is a line of a source document which is processed in such a way that it appears as an independent line in the output. Line elements can occur only within the bounds of an example or a figure body. There are no tags for a line element; one starts at the beginning of an input line and terminates at the end. A line element can contain text items.

Within the bounds of an example, elements that do not have explicit termination tags (**dd**, **li**, **lp**, **note**, **p**, and **pc**) are implicitly terminated at the end of the input line. Succeeding untagged input lines are treated as line elements. (As usual, the entered content may include **psc** elements to obtain special graphic effects, and .IM control words to imbed content from other files.)

If no content is entered, the APF will generate white space as specified by the **depth** attribute to provide room for artwork to be stripped in for reproduction.

The first part of the following example shows an example element with generated content. The second part is an example element whose body contains only line elements.

```
  .
  .
as seen in the following graph:
:xmp depth='6p6'
:exmp
:pc.The chart below, in contrast, demonstrates that. . .
  .
  .
:p.This can be demonstrated in a variety of ways, but perhaps it is
best illustrated by the following proof:
:xmp
(1) A * B = C
(2) C = A * D
(3) B = D
:exmp
:pc.But other arguments are equally valid . . .
  .
  .
```

Processing With the "gdocprof" Profile: The supplied APF generates the required space if **depth** is specified. Otherwise, the content is processed without justification, running together of input lines, or spelling checking. In either case, the output of processing is kept together, and is placed on the next page if there is insufficient room on the current page. Examples are indented at the left margin.

Results:

as seen in the following graph:

The chart below, in contrast, demonstrates that. . .

 .
 .

This can be demonstrated in a variety of ways, but perhaps it is best illustrated by the following proof:

```
  (1) A * B = C
  (2) C = A * D
  (3) B = D
```

But other arguments are equally valid . . .

PART TWO: GML PROCESSING GUIDE

This section explains how to process documents that contain GML markup. The material covered applies to all types of document; text entry and processing considerations which pertain only to a specific document type are discussed in the GML Markup Guide for that type. A summary of the material on processing is included for quick reference in "Appendix B: GML Processing Summary" on page 139.

GML documents can be created on any word processing or text editing system which is capable of communicating its files to an IBM System/370 computer for processing by SCRIPT/VS. The exact procedures for doing this will depend upon the system you use.

One system you can use is the Conversational Monitor System component of VM/370. Information on its use can be found in IBM Virtual Machine Facility/370: CMS User's Guide. Another IBM system which can be used for preparing SCRIPT/VS GML input is the Time Sharing Option (TSO) of OS/VS2 MVS. It is described in IBM OS/VS2 TSO Terminal User's Guide. You can also use a Program Product: IBM Advanced Text Management System II (ATMS II).[11]

Note: Consult your document administrator for the actual procedures for the systems in use in your installation.

[11] OS 5740-XX2, DOS 5746-XXG

SCRIPT/VS is invoked by use of the "SCRIPT" command. A complete description of the command and all its options can be found in the Document Composition Facility: User's Reference. This section provides only a brief summary of those options most likely to be used by markup editors.[12] Since the exact form in which you issue the command and options depends upon your system, examples of use and explanations are presented separately for the CMS, TSO, and batch processing environments.

SCRIPT COMMAND OPTIONS

The first parameter of the command is an identifier of the file or data set which contains the beginning (or all) of your document. For example:

```
  SCRIPT  mydoc
```

The file identifier is followed by any options you may specify.

Some available options are shown in the following list. The options (like the remainder of the command) can be entered in any combination of upper and lower case characters. In this list, the upper case letters signify acceptable abbreviations for the options.

Processing Options:

PROfile(name)	Specifies the name of a file or data set you want used as the document profile. The profile(s) available depend on the type of document you are processing, and are described in the GML Markup Guide.
SYsvar(x value)	Specifies variations of the normal processing for the profile in use. There can be one or more pairs of variation codes ('x') and corresponding values. The variations possible depend on the type of document being processed, and are discussed in the GML Markup Guide.
SPellchk	Causes SCRIPT/VS to perform spelling verification when requested with the .SV control word.
TWopass	Causes SCRIPT/VS to process your input document twice, so it can resolve forward references (as when an automatically generated table of contents is in the front of a book). The output is produced after the second pass. This option approximately doubles the computer processing time needed for a run.
Bind(n1 n2)	Specifies that the printed portion of the output page is to be shifted to one side to leave room for binding. It is shifted n1 space units to the right on odd pages, and n2 space units on even pages. (You would normally make n2 smaller than n1 for duplex printing, so even pages would have a large enough right margin for binding.) The n2 value can be omitted when you want all pages shifted the same amount (as you normally would for simplex printing).

Logical Device and Output Destination:

DEVice(type)	Specifies the "logical" device and underlying physical device for which formatting is to be performed. In this context, "logical" means a combination of physical device type, page size, and number of lines per vertical inch. The logical device types supported in the starter set are shown in Figure 19 on page 140. If you do not specify a device, "term" is the default in the CMS and TSO environments, and "1403W6" in the batch environment. The output destination of the document will be the physical device, unless the FILE, PRINT, or TERM options are specified.

[12] Your installation's procedures may require you to specify options in addition to those you choose from this manual.

File(name)	Causes output to be formatted for the specified logical device, but stored in the named file or data set instead of being displayed or printed.
PRInt	Causes output to be formatted for the specified logical device, but printed on the system printer. If no logical device is specified, "device(1403W6)" is assumed.
TErm	Causes output to be formatted for the specified logical device, but displayed at a terminal instead of on the physical device. If no logical device is specified, "device(term)" is assumed. Note: TERM is not available in the batch environment.

Error Handling:

Message(delay)	This option, with the parameter "delay," causes error messages to be printed at the end of the output document, instead of being displayed at a terminal during processing. (In the batch processing environment, messages are always delayed.)
COntinue	Normally, SCRIPT/VS stops processing when it finds an error. This option directs it to continue processing unless the error is a severe one.

ISSUING THE SCRIPT COMMAND

The command is discussed separately for each available environment: CMS, TSO, and batch processing. In the examples, option names are shown in upper case for emphasis.

Note: Consult your document administrator for the actual operating procedures for your installation. These could include combinations of system procedures (that is, execs, proclibs, or JCL) and manual operating procedures.

CMS

The Conversational Monitor System (CMS) component of VM/370 is a single-user operating system that provides interactive file creation and manipulation facilities that can be useful for document development. SCRIPT/VS uses the input/output facilities of CMS to read source documents and write the formatted output. Processing output may be directed to your CMS terminal, a VM/370 printer, or a CMS file for later use.

The APFs and a profile must be available to CMS before invoking SCRIPT/VS to process your document.

When issuing the SCRIPT command, you must separate the options from the file identifier by a left parenthesis. A default filetype of "script" is assumed for all files if a filetype is not specified.

In the following example:

```
SCRIPT techrprt ( PROFILE(gdocprof) TWOPASS BIND(.5i)
```

- "techrprt" is the filename of the document to be processed. The default filetype of "script" is assumed.
- The PROFILE(gdocprof) option causes SCRIPT/VS to use the file "gdocprof script" as the profile.
- The TWOPASS option tells SCRIPT/VS to make two passes through this file before formatting it for output.
- The BIND(.5i) option causes SCRIPT/VS to shift the printed output .5 inches to the right on all pages.

- The logical device for which the output would be formatted is "TERM," since the device option was not specified. The destination would be the same as the logical device, since no destination was specified. The results would therefore be displayed at your terminal.

In the next example:

```
SCRIPT mydoc ( M(delay) PROFILE(gdocprof) PRINT SPELLCHK
```

- "mydoc" is the filename of the document to be processed. The default filetype of "script" is assumed.
- The M(delay) option causes error messages to be appended to the output, instead of being displayed at the terminal during processing.
- The PROFILE(gdocprof) option causes SCRIPT/VS to use the file "gdocprof script" as the profile.
- The PRINT option tells SCRIPT/VS to direct output to the system printer. Since no logical device was specified, "device(1403W6)" is assumed.
- The SPELLCHK option causes SCRIPT/VS to verify the spelling of the words in your document when requested with the .SV control word.

TSO

The Time Sharing Option (TSO) is a processor available to OS/VS2-MVS users. SCRIPT/VS uses the input/output facilities of TSO to read source documents and write the formatted output. Processing output may be directed to your TSO terminal, an MVS printer, or a TSO data set for later use.

APFs and a profile data set must be available to TSO before invoking SCRIPT/VS to process your document.

All data set names which are not fully-qualified assume "userid." and ".text" as default qualifiers.

In the following example:

```
SCRIPT techrprt(rpt1) PROFILE(gdocprof) TWOPASS BIND(.5i)
```

- "techrprt" is the name of a partitioned data set whose member "rpt1" is the document to be processed. The default qualifiers of "userid." and ".text" are assumed, thus producing "userid.techrprt.text(rpt1)" as the fully-qualified name.
- The PROFILE(gdocprof) option causes SCRIPT/VS to use the file "userid.gdocprof.text" as the profile.
- The TWOPASS option tells SCRIPT/VS to make two passes through this file before formatting it for output.
- The BIND(.5i) option causes SCRIPT/VS to shift the printed output .5 inches to the right on all pages.
- The logical device for which the output would be formatted is "TERM," since the device option was not specified. The destination would be the same as the logical device, since no destination was specified. The results would therefore be displayed at your terminal.

In the next example:

```
SCRIPT 'userid.mydoc' CO PROFILE(gdocprof) PRINT SPELLCHK
```

- "userid.mydoc" is the fully-qualified data set name of the document to be processed.
- The CO option causes SCRIPT/VS to continue processing even after an error is found, unless the error is a severe one.

- The PROFILE(gdocprof) option causes SCRIPT/VS to use the file "userid.gdocprof.text" as the profile.
- The PRINT option tells SCRIPT/VS to direct output to the system printer. Since no logical device was specified, "device(1403W6)" is assumed.
- The SPELLCHK option causes SCRIPT/VS to verify the spelling of the words in your document when requested with the .SV control word.

BATCH PROCESSING

To process SCRIPT/VS documents in batch environments, the Document Library Facility (DLF) program product is required. Processing output may be directed to a system printer or to a document data set.

Information about DLF can be found in the _Document Library Facility Guide_. Consult your document administrator for information about the batch processing procedures in use in your installation.

This section will show you how to design GML markup to fit the exact needs of your installation. There are discussions of SCRIPT/VS functional capabilities and GML concepts to convey a better understanding of how the starter set works and what else the system can do for you. Material is presented which will enable you to:

- Define GML tags to describe your installation's documents.
- Specify APFs and profiles that must be developed to process them.
- Establish conventions and procedures for marking up and processing documents.

DEVELOPING GML FOR YOUR OWN USE

GML is customizable; it provides the syntax and usage rules for developing your own vocabulary of tags for describing the parts of a document. SCRIPT/VS recognizes GML markup and interprets it according to the profile and APFs that are specified for use during processing. SCRIPT/VS provides the symbol and macro facilities to create profiles and APFs.

As an illustration of how to apply these functions, IBM has created the starter set of GML described in Part One of this manual. You can use the same facilities to tailor the starter set, or to create entirely new sets of GML, for the different kinds of documents you produce (for example, reports, lists, catalogs, proposals, specifications, bulletins, manuals, directories, form letters, and computer generated reports). For each kind of document, you can define the GML tags that are most meaningful to the people who produce documents of that kind. And you can define document profiles and processing functions that will cause your style of printing to be produced for each kind of document. How you accomplish these things will depend on the role GML plays in your installation.

GML IN YOUR INSTALLATION

GML is a method for marking up documents in a way that describes the document. GML provides a syntax and conventions for markup, but does not require any particular vocabulary to be used. This is because the function of the GML language is to describe documents, and different kinds of document require different vocabularies to describe them. In other words, GML lets you create the precise vocabulary of tags you need to mark up your documents.

SCRIPT/VS is a system and language for processing documents. SCRIPT/VS allows you to define new interfaces to its function to suit your own needs. This process can be as simple as merely changing the behavior of a single SCRIPT/VS control word, or it can involve providing an entirely new markup language, such as GML, for the users of the system.

If you are using the SCRIPT/VS language directly, the kinds of changes you might want to make would probably be in the nature of adding a few special-purpose macros. When your installation uses GML, on the other hand, you will want to use the markup definition capability of SCRIPT/VS to refine the vocabulary of tags so that the language can be used to describe your own documents, using words that reflect how you actually view their structure and purpose.

To take full advantage of GML, some preparation is needed. The documents to be marked up should be analyzed, so that the vocabulary of tags needed to describe them can be established. Also, the actual composition or formatting for each described item must be defined and implemented. Sometimes, you may want to have several alternative styles of composition available for each tag. These need not all be defined at the start -- one of the benefits of having your documents marked up with GML is that you can later decide to process the same document in a way that was not foreseen at the time it was created and marked up. Since the markup of the document merely describes the document's characteristics, different processing is possible by simply writing new processing functions to be associated with the same descriptive tags.

For purposes of explaining what is involved in these tasks, and how to do them, this manual characterizes them as jobs performed by different individuals, including:

document administrator:	One who is responsible for defining markup conventions and procedures for an installation. This involves defining the actual vocabulary of tags to be used and also the nature of the processing required for each.
markup editor:	One who marks up individual documents according to an installation's conventions and procedures established by the document administrator.
text entry operator:	One who actually enters the text and markup into machine-readable form on a terminal or word processor.

text programmer: One who implements APFs that provide the processing specified by the document administrator. In SCRIPT/VS, this involves writing SCRIPT/VS macros and organizing macro libraries and profile files so that the appropriate composition will be done for each tag.

Of course, all of these roles could be performed by the same person. And, particularly in larger installations, many persons could perform each of these roles. Some of the roles might be performed by persons who have other responsibilities as well. Authors, for example, might also perform the tasks of a markup editor and text entry operator. If you are developing your own informal documentation, you might perform all four roles in addition to being the author. For the sake of clarity, therefore, each role is discussed in this manual as if it were performed by a separate individual.

THE ROLE OF DOCUMENT ADMINISTRATOR

To perform the document administrator role, you must be familiar with text processing and with basic programming concepts (although you need not be a programmer). You must know those of your installation's documents for which you are responsible, and the activities associated with them. These activities not only include publishing production, but such auxiliary activities as cataloging, indexing, and the like, since SCRIPT/VS and GML can assist in those as well.

You will design the markup for your documents, and you may create one or more "GML Markup Guides" that show how to prepare the documents for processing. Part One of this manual is an example of such a guide. It is frequently cited as an example in this Part, where it is referred to as the "Markup Guide."

As document administrator you will specify the processing required for your documents. You therefore need to know the functional capabilities of SCRIPT/VS. (You need not know the details of coding control words or SCRIPT/VS macros since the actual implementation of the processing is done by the text programmer.) In particular, you should be well acquainted with the symbol and macro capabilities, as these are very important in GML support.

These subjects are introduced in the next chapter. However, a more detailed knowledge than is provided in that chapter is sometimes required for other discussions in this Part. You will then need to refer to the IBM Document Composition Facility: User's Guide for further information.

Note: You may also need to assist text entry operators and to organize and manage your installation's document data base. These activities may require knowledge of the text entry and edit systems used by your installation, the word processing systems, the output devices (including such characteristics as fonts and paper sizes), and the facilities for data base management. Because these areas vary greatly among installations, the subject is not addressed in detail in this manual.

SCRIPT/VS FUNCTIONAL CAPABILITIES

The starter set APFs utilize many of the capabilities of SCRIPT/VS. From the discussions of processing in the Markup Guide, you are aware that SCRIPT/VS formats text for printing on impact printers and on nonimpact printers, such as the IBM 3800 Printing Subsystem. With the 3800, SCRIPT/VS can format text in multiple character styles and sizes (called "fonts"). Thus, for many documents, SCRIPT/VS can provide flexible composition for printing on a computer printer as an alternative to using independent composition machines or sending typesetting jobs to a vendor.

SCRIPT/VS can also be used as a "preprocessor" to prepare documents for processing by other programs, such as formatters that support photocomposers.

This chapter summarizes SCRIPT/VS functional capabilities so you will know what your own APFs can be made to do. User-controllable SCRIPT/VS processing includes two broad categories of function: formatting, and general document handling.

FORMATTING FUNCTIONS

The formatting functions provide:

Page Layout: You can control page dimensions, the number of columns per page (up to nine), running headings and footings, and single or double spacing.

Page layout includes:

- Line Formatting. You can control how input lines are processed for output: formatted or unformatted; centered, aligned left, or aligned right; justified (so that output lines are the same length) or ragged-right.
- Spacing. You can control the amount of space left between output lines, including the reservation of space for illustrations to be stripped in.
- Paragraphing. You can control the style of paragraphing (spacing between paragraphs and indention).
- Fonts. When using the IBM 3800 Printing Subsystem, you can control which fonts are used for different portions of text, both in the body and in running headings and footings.
- Columns. You can define the number of columns as well as the size of each one and its placement on a page.
- Margins. You can control the size of the top and bottom margins as well as the left and right margins. Title lines can be defined that will be put into the top and/or bottom margins.
- Indention. You can have indention done in a number of ways. For example, you can have hanging indentions, left or right margin indention, and single line indentions.
- Headings and Footings. You can have running headings and footings with or without page numbers, and separate treatment for odd and even pages.

Head Levels: You can specify up to seven head levels for distinctive formatting of headings for different levels of topics. Distinctive formatting includes before and after spacing, font selection, capitalization, underscoring, and right or left alignment. For example, the heading of this chapter has a head level of 1. The run-in heading of this paragraph has a head level of 6.

Table of Contents: You can control whether or not a table of contents is automatically generated and where it is placed. SCRIPT/VS collects entries for a table of contents from the text of head levels and supplies the page number. You can also specify phrases other than the text of head levels to appear in the table of contents. The table of contents of this manual was automatically generated by SCRIPT/VS.

Highlighting Phrases: You can control how phrases are to be highlighted for emphasis. For devices that don't have multiple fonts, highlighting is done with underscore, uppercase, or uppercase underscored. For devices that support multiple fonts, you can change font for emphasis.

Footnotes: SCRIPT/VS saves text indicated as a footnote and places it at the bottom of the page.[13] Subsequent footnotes are placed below it.[14]

Hyphenation: You can control whether or not words are to be hyphenated at the end of output lines. SCRIPT/VS provides a dictionary of more than 10,000 English words. For hyphenation, SCRIPT/VS looks in the dictionary to determine how a word is broken up into syllables and applies an algorithm for prefix and suffix variations. (This algorithm significantly extends the basic 10,000.) You can also add words to a supplementary dictionary for possible hyphenation in a particular document.

Printing of Portions of the Output Document: You can control whether every page of formatted text is put in the output document or only the range or ranges of pages specified.

Tab Handling: You can specify the setting of tabs. When formatting output lines, SCRIPT/VS tabs to the right to the prescribed tab stop.

Box Drawing: You can construct boxes around text (which can still be formatted in the usual ways). You can also draw boxes within boxes and can draw vertical lines to separate columns of text within a box and horizontal lines to separate rows.

Keeping Text Together: Standard SCRIPT/VS processing includes a way of keeping text together to improve the appearance of output. For example, SCRIPT/VS keeps the text of a head level and the following three lines of output text together, so that they appear in the same column. You can specify other types of text to be kept together.

Use As A Subroutine: In a batch environment, with the IBM licensed program Document Library Facility installed with SCRIPT/VS, an application programmer can code programs to call SCRIPT/VS as a subroutine (for example, to format reports).

GENERAL DOCUMENT HANDLING FUNCTIONS

The general document handling functions can be used in conjunction with formatting, or by themselves. They include:

Saving Input Lines for Subsequent Processing: You can control whether certain input lines will be written to a data set or file.

Revision Codes: You can control the placement of up to nine distinct revision codes in the left margin to indicate lines revised since a previous version of the document. (Revision codes may be used to flag lines for other reasons as well.)

Spelling Verification: You can control whether or not words are checked for spelling. SCRIPT/VS uses the same dictionary and English-language prefix and suffix algorithm for spelling verification that it does for hyphenation. The additional words that you can specify for a particular document for hyphenation can also be used for spelling verification.

Imbedding of Separate Files: You can control how separate source documents are brought together for processing as a single document. Any number of source documents can be imbedded in the base source document. A source document that has been imbedded can itself have another source document imbedded in it, and so on, up to eight levels.

Symbol and Macro Instruction Processing: You can define symbols and macro instructions for substitution during processing. Symbols have many uses: for example, in tests for conditional processing, for cross-references to pages or figure numbers, for entering characters unavailable on the entry keyboard, and as abbreviations for lengthy, repetitive phrases. You can define what a particular macro instruction will do. For example, you might redefine a particular head level to alter the SCRIPT/VS formatting style. The symbol and macro instruction facilities

[13] Up to 10 output lines per footnote.
[14] Like this.

are used to implement the Generalized Markup Language described in this manual.

Conditional Processing: You can cause SCRIPT/VS to test for user-controllable conditions that can alter processing. For example, the symbol values set by the user might determine whether a block of input text is included in the output document or not. (SCRIPT/VS includes much conditional testing as part of its normal processing. For example, it checks the amount of space left in a column before processing certain blocks of text. Much of this conditional processing can also be controlled by the user by defining macro instructions to supplement SCRIPT/VS control words.)

Personal Interaction During Processing: In an interactive environment (CMS or TSO), you can affect SCRIPT/VS as it processes by entering text and/or markup at a terminal. In effect, the terminal can be treated as an input file. For example, the user can interactively define the values of symbolic variables specified in the document or enter those portions of text that vary from one processing time to the next.

Destination of Output: You can control whether the output document is, (1) stored as a file for possible later editing, printing, or as input to another program (for example, a formatter that supports a photocomposer). Or, (2) printed on the device specified, which includes both impact and nonimpact printers and display and typewriter terminals.

Tracing of Processing Actions for Debugging: You can trace all control words and each step of symbol and macro substitution in input lines. In cases where unexpected output occurs as the result of executing a macro, which may contain many control words, trace information can be an invaluable aid to pinpointing the problem area.

The Markup Guide shows how to use GML markup, but says little about the benefits to an installation of doing so, or the way in which SCRIPT/VS actually interprets the markup. When you design your own GML markup, you will make decisions that depend (among other things) on the GML benefits you wish to obtain, and the markup interpretation capabilities available. These subjects are introduced in this chapter.

OBJECTIVES OF GML MARKUP

SCRIPT/VS functions are invoked by "marking up" your source document. To mark up a source document is to add information to it that enables a person or system to process it in some way. In SCRIPT/VS, the added information, or "markup," can be control words, or it can be GML tags that describe the characteristics of a source document without respect to particular processing.

Although some SCRIPT/VS control words deal with general document handling functions and symbolic representation of data, the control words that usually come to mind when speaking of document markup are those that control the actual composition of the document. For example, control words would allow you to specify that text should be set in a column 25 picas wide. This composition control would always have the same meaning, independent of the content of that column.

GML markup, on the other hand, has an entirely different purpose. GML is a descriptive language -- with GML you describe the document. For example, GML markup allows you to identify that certain text is a paragraph, or that other text is part of a numbered list. That paragraph will always be a paragraph in this document, independent of how it is formatted in a particular run. The author of a document can specify this kind of markup without understanding the composition or publishing conventions that will be applied. He can concentrate entirely on the content and purpose of the document, which is his own field of expertise.

When a document is marked up with GML, it can be processed in various ways, merely by providing different interpretations for the same set of tags. These interpretations can result in different composition, or they can even result in processing other than formatting.

GML WITH SCRIPT/VS TEXT PROCESSING

Here are some benefits in using GML when you use SCRIPT/VS only for text processing:

- Alternative GML interpretation. A GML tag need not be limited to a single SCRIPT/VS interpretation. For example, a tag might indicate that a group of words in the text is a paragraph. For one application, you might want the first line of a paragraph to be indented. But for another application, you might want paragraphs to be set in block style, with spaces between them. Each application can be satisfied by alternative GML interpretations, with no change to the source document or to the markup of paragraphs. You have control over the way GML is interpreted.

- Ease of markup. It is easy to remember GML tags because they can consist of terms and abbreviations commonly used to describe a document. GML generally requires fewer characters to be entered for markup than a corresponding complex sequence of control words. The result is faster markup and keying of the document. Changes to the markup are also faster and may even be eliminated with GML, because you can control how GML is interpreted.

- Ease of text update. It's easy to update text marked up with GML. With GML, such things as the numbering of items in an ordered list is usually left to the formatter, which numbers the items automatically. Thus, when you insert or delete an item, you don't have to renumber subsequent items, as you would have to if the numbers were part of the source text.

- Uniformity of formatting style. Use of GML for all documents of a particular kind results in a single format for all those documents -- without the persons that do markup even having to think about format. Similarly, a change in format for all the documents could be achieved simply by changing the way GML is interpreted -- without anyone having to be retrained to do markup in a different way.

GML WITH OTHER APPLICATIONS

For applications involving general processing with SCRIPT/VS, or other programs, the benefits of GML include:

- Alternative text processing programs. SCRIPT/VS could be used to interpret GML into the processing controls of text processing programs other than SCRIPT/VS. For example, they might be interpreted into the controls of a composition program (such as the IBM Installed User Program TERMTEXT/Format, 5796-PBR), which could be used to produce output for a photocomposition device.

- Document exchange. Documents with GML can be processed by different groups or locations more easily than documents with the specific markup of a particular "house style." With GML, each user can obtain his own formatting style, and support his own output devices, by using his own interpretation of the tags.

- Data-base applications. GML describes the contents of documents, so that programs can identify information in them. Programs could be written alone, or in conjunction with SCRIPT/VS processing, to extract information from documents for data base construction or to retrieve information for data base sharing.

- Unique customer applications. Precisely because GML is general, it allows for numerous applications unique for a particular customer. GML designed for this purpose could, for instance, be used both for formatting printed output and for information retrieval.

HOW SCRIPT/VS INTERPRETS GML

In GML, "interpreting" a tag means performing the correct processing function on its associated text. The choice of processing functions will depend upon the desired application. (Processing functions are called "application processing functions" or "APFs.") For example, obtaining a proof copy of a document on a terminal would be considered a different application from obtaining final copies on an IBM 3800 Printing Subsystem. In each case different processing functions for some of the tags would be used. The tags would not change -- only the associated APFs.

In SCRIPT/VS, APFs are implemented as sets of control words. These sets of control words need not be placed in the document itself. They can be placed in a separate document called a "profile." SCRIPT/VS processes the profile before it processes the document marked up in GML. Profiles for different kinds of applications can have different APFs for some tags, but use common APFs for others.

SCRIPT/VS has still greater flexibility in that each APF can also be defined as an individual document. These APFs are defined in documents separate from the profile and stored in host system libraries, so they will be available for the processing of many documents and applications. Figure 7 on page 103 shows a source document and a profile with two APFs established as separate documents.

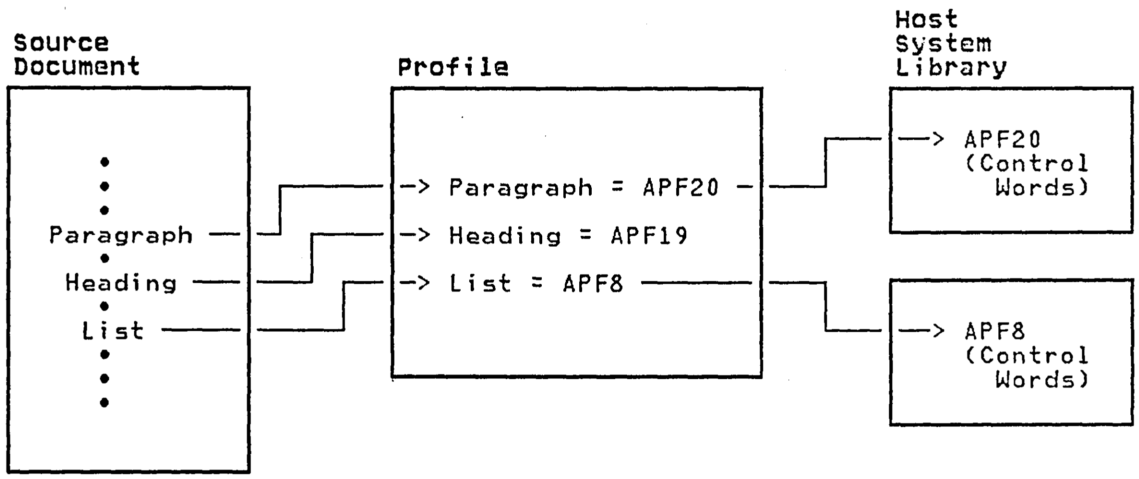

Figure 7. Document and Profile with APFs in Separate Documents

You may want a single APF to operate on the text associated with many different tags, or a single tag's text to be operated on by a number of different APFs for different applications. To accomplish this, SCRIPT/VS provides a way of associating a tag with the name of the APF that is to act on its text. As SCRIPT/VS processes a document marked up with GML, it interprets GML tags by associating them with APFs.

One or more tags can be associated with the same APF. This will cause the same processing to occur for the text associated with each tag. Figure 8 shows this relationship.

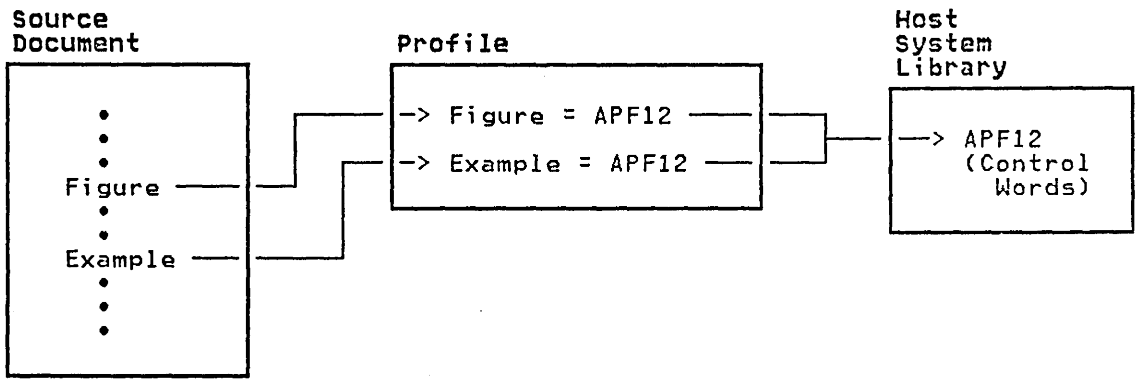

Figure 8. Different Tags Processed by the Same APF

Figure 9 on page 104 illustrates the use of multiple profiles. When you run SCRIPT/VS, you indicate the profile for the application being run. The profile then associates each tag with the correct APF for that application.

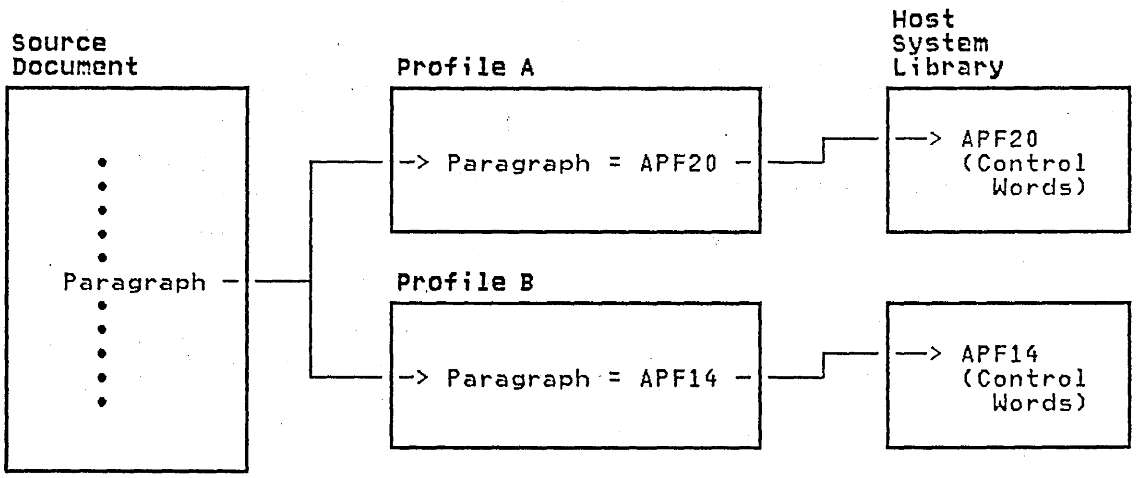

Figure 9. Multiple Profiles for Different Applications

SCRIPT/VS provides the facilities for you to define the profiles and APFs as you want them. Profiles and APFs are just documents with special uses. You create them, update them, and store them the same as your text documents.

To specify APFs and profiles, you must first know the characteristics of your documents. This is because a document's characteristics are what cause you to want to process it in a particular way. Such formatting, for example, as beginning chapters on a new page, italicizing emphasized phrases, and indenting lists, is all done to assist the reader's comprehension by emphasizing the structural characteristics of the document.

GML, plus the vocabulary of markup tags you define, form the means of telling a processing system about the characteristics of a type of document, so the correct APFs can be associated with them. But before you can define these tags, you must know the document characteristics the tags will identify. Determining the characteristics of a type of document and recording them in an unambiguous fashion is called "creating a document type description."

DOCUMENT CHARACTERISTICS

A document has two primary characteristics, one being its role or purpose. This characteristic is called the document "type."[15] It is literally an answer to the question: "What type of document is this?" The other primary characteristic of a document is its content.

GML tags serve to identify the structure of the content of a document. That is, they identify the document's component parts and their relationship to one another. These component parts are known as elements. An element has the same two primary characteristics as a document: type and content. In a typical document, you find element types such as paragraph, chapter, list, heading, and so on. In a specialized type of document, such as a service manual, you might find such specialized element types as "parts list," "removal instructions," and "replacement instructions." Whether common or specialized, elements of the same type are always processed identically to one another.

The elements of a document are organized in a hierarchy: typically, the content of a chapter would include element types such as list and heading; the content of a list would include such lesser elements as list items and list parts; and so on. At any point in the hierarchy you could (in theory, at least) remove an element and process it by itself; while you did so, that element would be a document. A document, then, is just an element that is at the highest level of the hierarchy. (Or, you could say an element is a document that is within the hierarchy of some larger document.)

A document (or an element) can have secondary characteristics in addition to the two primary characteristics of "type" and "content." These secondary characteristics are referred to as "attributes" when it is necessary to distinguish them from the primary characteristics. One example of an attribute is the shortened version of a heading or title (**stitle**) which can be used instead of the full version in running headings or footings. We saw other examples in the Markup Guide.

The type, content structure, and attributes must all be reflected in your document type descriptions. The way in which you prepare the descriptions can have a significant effect on the ease with which your documents can be marked up, the variety of applications which can be performed on them, and the accuracy with which APFs and other processing can be specified.

Without an accurate and complete document type description, it is difficult to tell markup editors precisely where in a document they can insert particular tags. Nor can a text programmer be certain of the state of processing that will exist when his APFs are executed.

Fortunately, development of document type descriptions need not be a formidable task. Sometimes they already exist, in the form of style books, outlines, or procedure manuals. Whether or not you have such aids available, you begin by analyzing your installation's documents. This involves three activities:

[15] It is also known as the "GML type," because of the importance of "type" in GML markup.

1. Determining document and element types;

2. Defining document and element structure; and

3. Defining attributes.

For clarity, these activities will be discussed as if they were performed in sequence. In practice, there is significant interaction among them; at times you will perform them simultaneously.

DETERMINING DOCUMENT AND ELEMENT TYPES

The first step is to examine your installation's documents and group them by type. There may be some classes of specialized document which have unique characteristics, and which are produced often enough that it is clearly worth categorizing them as unique document types (for example, quarterly editions of a phone directory, periodic revisions of a procedure manual, standard contracts, etc.).

At the other extreme are documents which are neither specialized nor numerous, and which can be grouped together as a single "generic" document type. The Markup Guide describes a markup design for such documents, which it calls "General Documents." Most of the elements of a General Document are generic in the sense that an element's type suggests only the structural role the element plays in the document, rather than the semantic role as well.

Compare the General Document "ordered list," for example, with a "removal instructions" element that might be part of a "service manual" document type. Both suggest a numbered list, but the second tells what it is a list of. Many more elements could be identified as ordered lists than as removal instructions, but doing so would preserve less information about those elements.

Most documents will fall somewhere between the extremes of "clearly specialized" and "clearly generic." As an example, consider the Markup Guide itself. While most of its elements could be treated as General Document elements with little loss of GML benefits, the individual tag descriptions in the section "Element Tag Descriptions" on page 21 present a unique situation. Figure 10 shows how a tag description could be viewed as part of a General Document.

```
    head three (H3)
    markup rule (graphic PSC)
    paragraph (P)
    fifth-level heading segment (implied by H5)
    example (XMP)
    fifth-level heading segment (implied by H5)
        head five (H5)
        basic document elements
        sixth-level heading segment (implied by H6)
```

Figure 10. Structure of a Tag Description (Generic)

In contrast, Figure 11 on page 107 shows the structure of a tag description when it is treated as a specialized element.

Note the differences between the two approaches.

- The specialized view shows more detail about the tag description.

  For example, we know what occurs within the markup rule. Similarly, the division of the description into three parts (element type definition, tag usage discussion, and processing discussion) is identified. This could be a factor in processing: you might specify that the APFs should keep each part on the same page.

- The generic view uses fewer tags, and appears simpler, but it is largely inaccurate.

```
    element tag description (implied by TYPE)
        element type definition (TYPE)
            markup rule (implied by TYPE)
                element tag (implied by TYPE)
                attribute labels (ATTR)
                additional attribute labels (ATTRMORE)
                content rule (CONT)
                termination rule (TERM1)
                termination rule continuation (TERM2)
            element description (DESC)
            attribute description (ADESC)
        tag usage discussion (implied by USAGE)
            usage rule (USAGE)
            tag usage example (TXMP)
        processing discussion (implied by PROC)
            processing specification (PROC)
            processing example (PXMP)
```

Figure 11. Structure of a Tag Description (Specialized)

Some of the tags do not really describe the elements, although they are associated with APFs that will produce acceptable formatting. For example, the tag description is not a third-level heading segment, although it begins with a head three. In a General Document, as defined in the Markup Guide, a head three always begins a third-level heading segment. Similarly, in the tag description the first fifth-level heading segment must be followed by an example (**xmp**). This is not true of General Documents.

These considerations suggest strongly that specialized tags should be used for the tag description section. (Frequency of production of the document is less significant here, since even in a single GML Markup Guide there will be dozens of tag descriptions.) However, let us go a step further and mark up a tag description using both starter set and specialized tags. For the specialized tag case, we will also define some attributes, and some symbols for repetitive text. (Repetitive text is sometimes called "boilerplate.")

Figure 13 on page 108 shows how the tag description "h1 -- head one" on page 55 could have been marked up using only tags and symbols in the starter set, while Figure 12 illustrates the use of specialized GML.

```
:type id='tdh1' tag='h1'.head one
:attr.id= stitle=
:cont.&osl.
:term1.&ae1.
:term2.&cpl.
:desc.a first-level heading ... called a :q.chapter:eq..
:adesc atag='id' aname='unique identifier'.&adid.
:adesc atag='stitle' aname='short title'.an alphameric ... heading.
:usage.within the ... back matter.
:txmp
   .
   .
&gml.h1 id='mupro'.Markup Procedures
&gml.p.This section explains:
   .
   .
:proc prof='gdocprof'.begins ... specified.
:pxmp.:hp1.The heading :hdref refid='mupro'. is an example.:ehp1.
```

Figure 12. Element Tag Description Marked Up With Specialized GML

The two markup examples produce results that are almost identical; differences are within the normal stylistic variations of APFs and profile options. Nonetheless, there are significant differences between them.[16]

```
:h3 id='tdh1'.h1 -- head one
:psc
.cm graphic
.cm to obtain special formatting of markup rule
.tb .2i  1.2i 3.0i 5.1i 7.2i
.if &$PDEV eq 3800 .tb .2i  1.2i 2.8i 4.6i 6.6i
.sk 3
.if &$PDEV ne 3800 .bx .2i  1.2i 3.0i 5.1i 7.2i
.if &$PDEV eq 3800 .bx .2i  1.2i 2.8i 4.6i 6.6i
.fo off
&$tab.Type&$tab.Attributes&$tab.Content&$tab.Termination
.bx
&$tab.&gml.h1&$tab.id= stitle=&$tab.Text on Same Line&$tab.At End of Line
&$tab.&$tab.&$tab.&$tab.See "Usage" in text
.bx off
.sk 2
.fo on
:epsc
:p.The :hp2.h1:ehp2. tag identifies
a first-level heading ... called a :q.chapter:eq..
:p.The :hp2.id:ehp2. attribute label identifies the
optional unique identifier attribute.
Its value is a string of up to five alphameric characters which can be
used to refer to this element.
No other head one can have the same unique identifier.
:p.The :hp2.stitle:ehp2. attribute label identifies the
optional short title attribute.
Its value is an alphameric ... heading.
:h5.Usage
:p.The head one can occur
within the ... back matter.
:xmp
   .
   .
&gml.h1 id='mupro'.Markup Procedures
&gml.p.This section explains:
   .
   .
:exmp
:h5.Processing With the "gdocprof" Profile
:p.The supplied APF
begins ... specified.
:h6.Results:
:p.:hp1.The heading :hdref refid='mupro'. is an example.:ehp1.
```

Figure 13. Element Tag Description Marked Up With Starter Set

- The generic case requires many more keystrokes. Substantial boilerplate must be entered which, in the specialized case, is generated by the APFs and symbols.

- The graphic **PSC** element in the generic case is application-dependent and system-dependent. Fortunately, most of it is identical for each tag description, and can be duplicated by using an imbedded file or macro, or the duplication functions of a text entry system.

  However, the two lines between the .BX and .BX OFF control words must be keyed separately for each tag description, and they are rather complex to enter. The corresponding specialized tags (**attr**, **attrmore**, **cont**, **term1**, and **term2**), on the other hand, are simple to enter. They can be associated with APFs other than those which format the tag description, such as the APFs which automatically generated the tag summary table in "Appendix A: GML Markup Summary" on page 131.

[16] Some text is replaced by ellipses (...) to keep the examples short; this text is identical in both cases. The starter set example uses the system symbol "&$tab" instead of the tab key character, which would appear as a blank when printed.

- In the specialized case, the three parts of the tag description are kept together. (See "fig -- figure" on page 38 for an example.)

- The starter set markup will require a **psc** element whenever the processing example element requires a live example (rather than a reference to one, as was the case here).

- The specialized markup creates a far superior "document data base." Each element of a tag description can be located individually, and specialized processing defined for it. In the generic case, in contrast, you cannot tell the **h3** and **h5** elements from any other **h3** and **h5** that might be in the document.

  One practical application of this distinction was the generation of the tag summary table mentioned above. Another might be the documentation of a particular profile by printing only the processing specification elements (**proc**) with a designated **prof** attribute.

The benefits of using specialized tags must be weighed against the markup design and APF development costs, documentation of markup practices, training of markup editors, and possible need to conform to a document interchange standard.[17] For this manual, specialized tags were used for the tag descriptions, but the remainder was marked up as a General Document (with a few exceptions). Since the document as a whole did not conform to the description of a General Document, its document type was designated as "GML Markup Guide," with the tag **gmgdoc**. A description of the markup used can be found in "Appendix F: Document Type Description for GML Markup Guides" on page 148.

A significant point made by these examples is the importance of the initial approach to the problem. In Figure 10 on page 106 we started with a given vocabulary of tags and tried to make it fit the characteristics of a document. This resulted in inaccurate "force fits," but economized on tags and APFs. In Figure 11 on page 107, on the other hand, we started with the document and tried to let its characteristics determine the tags to use. This could have (although it did not in our example) lead to unnecessary proliferation of tags and APFs. This experience suggests the following approach to identifying document and element types:

1. Examine the document (or element) and attempt to identify its elements using the terminology that authors, editors, and readers use to describe them.

2. Roughly specify the processing you anticipate for each element.

3. Compare the tentative element types and processing to tags and APFs already in use in your installation, and decide whether new ones are really required.

This procedure must be done iteratively, and in conjunction with the other aspects of creating a document type description. Later in this chapter we will suggest some additional criteria to use in determining document and element types.

DEFINING DOCUMENT AND ELEMENT STRUCTURE

In order to prepare markup guides for your markup editors, and to fully specify APF processing for the text programmer, you must have a complete understanding of the structure of the type of document for which you are creating a type description. For each element, you must be able to answer the following questions:

1. What is the tag for the start of the element, or is it implicit in the start of some other element?

2. Does the element have attributes? Are the attribute labels mandatory or optional?

3. What is the structure of the content?

   a. Which types of element can occur in the content?

   b. In what order can they occur?

[17] The latter point can be handled by APFs that convert from one form of markup to another. See "Interchange Considerations" on page 127.

   c. Which content elements can occur more than once (are repeatable)?

   d. Which content elements are optional?

   e. Are there alternatives for any of the content elements?

4. How does the element terminate?

This section will suggest some techniques for representing the answers to these questions in a concise manner in "formal definition" diagrams. Such diagrams use a combination of tags and special characters to show the answers to structural questions.

One possible notation for formal definitions is illustrated in Figure 14, which shows the structure of a General Document as a whole.[18]

```
:gdoc sec=?.frontm, body, appendix?, backm? :e
```

Figure 14. Example of Formal Definition Notation

This figure answers the structural questions as follows:

1. A General Document is explicitly tagged with the tag "gdoc."

   This is shown by the use of the normal tag markup syntax, including the GML delimiter (:). (If the element were recognized implicitly, the delimiter would have been omitted.) For conciseness, tags and labels are used throughout the diagram instead of the full name of the element type or attribute.

2. A General Document has the attribute "security level"; its label is optional.

   Attributes are denoted by the normal attribute label markup syntax. The question mark signifies that the attribute label is optional; that is, it will not necessarily be entered for all elements of this type (i.e., General Documents) since a default value can be assumed if the label is omitted.

3. A General Document contains

   - an element whose type is "front matter,"
   - followed by an element whose type is "body,"
   - followed optionally by one whose type is "appendix section,"
   - followed optionally by an element whose type is "back matter."

   The period separates the markup from the content, just as it does in a source document when the content is immediate text.[19] The comma signifies a sequential relationship ("followed by") between the elements before and after it. The question mark denotes an optional element, just as it denoted an optional attribute label. In other words, some General Documents will not have an appendix, some will not have a back matter, and others will have neither.

4. The termination of the General Document is marked by an explicit tag.

   This is denoted by the ":E," which is the starter set convention for delimiting terminating tags.

The structure of the content elements can be shown in similar diagrams. Figure 15 on page 111 shows the front matter and body, and introduces additional notation techniques. The lines are numbered to facilitate reference.

[18] The full notation is summarized in "Element Types and Structure Definition" on page 145.
[19] The period could have been called the "content start" character, except that the Markup Guide requires that you begin a new line for each tag (except text items). This makes it unnecessary to delimit the content of an element except when it is immediate text. It would, in fact, not be incorrect to enter a period after every tag (or after the attributes, if the element has them), but these would be unnecessary keystrokes.

```
1.  :gdoc sec=?.frontm, body, appendix?, backm? :e
2.  :frontm.titlep, abstract?, preface?, toc?, figlist?
3.  :titlep.title, docnum?, date, author?*, address?* :e
4.  :address.aline* :e
5.  :abstract :preface.bde?*, hseg6?*, hseg5?*, hseg4?*, hseg3?*, :hseg2?*
```

Figure 15. Formal Definition of the Front Matter

Line 1 repeats the overall structure of the General Document. Since the content elements are displayed horizontally (rather than vertically, as was done in the informal structure diagrams in the Markup Guide), tags on the same line literally represent elements on the same level in the document's hierarchical structure.

Line 2 shows that the front matter is explicitly tagged, and has no attributes. Its content consists of a title page, which is mandatory, followed optionally by an abstract, a preface, a table of contents, and a list of illustrations. Some General Documents will not have one or more of the last four types of element, but all General Documents will have a title page.

No termination is shown for the front matter. This means it is terminated implicitly by the next element at the same or higher level. Line 1 shows that the body is the next element at the same level. Since the body is mandatory, we can conclude that the front matter can only be terminated by the body.

Line 3 describes the title page, which has explicit start and termination tags, and no attributes. It contains elements of five different types, of which all but the document title and document date are optional. The asterisk after the author tag indicates that it is repeatable. In other words, some General Documents could have more than one author element in their title page. Since author is also optional, the title page could have one author, many authors, or none.

Note that the address is also optional (and repeatable). However, Line 4 shows that its content contains one or more address lines, one of which must always be present when the address itself is. In documents which have no address element, there will of course be no address lines either, since they can occur only within an address.

The title page elements are separated by commas, which indicates that they must be entered in the specified order. If the commas were omitted, the title page elements would form a "set," and could be entered in any order. (The starter set APFs actually support a set structure, but the title page was defined as a sequence because it is easier to find things in a document prepared by another person when the entry order is prescribed.)

Line 5 introduces the notation for elements with identical structure. You can emphasize the commonality (and also save space in the diagram) by showing the elements on the same line. The markup for each element is shown on the left, separated by blanks. After the last piece of markup, the period and the definition of the content occur.

Here, the abstract and preface have an identical structure, which is much freer than that of the front matter. That is, there can be much more variation among different documents. The structure consists of optional basic document elements ("bde"), followed by optional heading segments ("hseg") of level 2 or lower, in ascending order.[20] The first-level heading segment, shown in Line 3 of Figure 16 on page 112, has an almost identical structure, differing only in that it begins with a mandatory head one.

Note that this definition permits successive headings without intervening text, since the basic document elements are optional. If your house style prohibits successive headings, you can reflect this in your document type descriptions by making the basic document elements mandatory.

[20] Ascending order is the only one possible, since a lower-level segment would be considered part of the content of a higher-level segment which preceded it.

```
1.  :body.part*/hseg1*
2.  part.h0, bde?*, hseg1*
3.  hseg1.h1, bde?*, hseg6?*, hseg5?*, hseg4?*, hseg3?*, hseg2?*
```

Figure 16. Formal Definition of the Body

Line 3 of Figure 16 is also an example of an element which does not require an explicit tag at the start. This is shown by the lack of a GML delimiter (:). The mnemonic name "hseg1" is not really a tag (since it is not used in markup), but is only a "token" used in formal definitions. In actual markup, the first-level heading segment is recognized by the tag for the head one (**h1**), which is the first element of its content.

Line 2 of Figure 16 shows that the part element has a more rigid structure than the first-level heading segment and the abstract and preface. Here, only the next-lower level of heading segment is permitted, preceded optionally by basic document elements.

Line 1 of Figure 16 introduces a notation (/) for expressing that a structure might permit a number of different element types to occur at a certain point. Here, the **body** can be one or more parts or one or more first-level heading segments (called "chapters" within the bounds of the body).[21]
Parentheses are frequently used with alternatives to show grouping (although they may be used wherever needed to resolve ambiguities).

Alternatives and grouping are also used in the following definition of the example (**xmp**) element:

```
:xmp depth=?.space/(line/bde)?* :e [no fig, fn, xmp]
```

It says that the content of an example either contains a space, or it contains optional intermixed line elements and basic document elements (literally, "a line or a basic document element, occurring zero or more times").

The brackets indicate a restriction expressed as narrative text, which is frequently less cumbersome than expressing the restriction formally as part of the definition. Here, the restriction is that figures, footnotes, and other examples cannot occur within the bounds of an example.

This discussion has covered the notational techniques you will need to create formal definitions. As a further example, a complete definition of the General Document is given in "Appendix E: Document Type Description for General Documents" on page 145.

But whether you use these techniques -- or whether you simply create your descriptions with narrative text -- is not very important. What _is_ important is that you ask the structural questions, determine the answers, and record those answers in a manner that leaves no ambiguities.

DEFINING ATTRIBUTES

Document type descriptions must indicate which elements have attributes, the attribute labels, and the permissible values for each attribute. One design problem which can occur is deciding when an element's characteristics should be treated as attributes, and when they warrant defining a new element type.

[21] It may seem confusing to say that a body must contain either parts or chapters, since the parts themselves consist of chapters. However, we use the word "contain" consistently to refer only to the first level of an element's content. When speaking of _all_ levels we use the phrase "within the bounds of" the element. Therefore, while a body that contains parts does not _contain_ chapters, chapters do occur _within the bounds of_ such a body.

[22] The tags for this are also included in the starter set.

In the starter set, for example, there is an element type of "highlighted phrase (type 1)." This also could be thought of as an element type of "phrase," with an emphasis or "highlight" attribute whose value is "1."[22] The choice of which view to adopt will depend upon such considerations as the ease of entry and modification, the clarity and meaningfulness of the marked up source document, whether you intend to define and support other attributes of a phrase, and whether "highlight" is defined as an attribute of other element types.

The elements **figref**, **fnref**, and **hdref** illustrate a similar situation. The content of these three types of element is empty. However, all three have a "reference identifier" attribute whose value is the unique identifier of the element being referred to. One could just as well decide to eliminate the **refid** attribute and enter the reference identifier value as the content of the element.

The title page element represents a different situation. Its elements can be thought of as attributes of the document as a whole. However, since some of them (such as "address") have structures more complicated than the simple strings that SCRIPT/VS permits in attribute values, it was necessary to find some other way to represent them. In the starter set, these attributes were such that they could be considered elements of the title page, but this would not work for all attributes. One approach is to define an element to be a "holder" for attributes. Such an element would not be part of the sequential content of the document. (In this respect it is similar to the footnote, which also does not appear in output at precisely the point at which it is entered.) Its elements would be considered to exist along with the document for the duration of processing. In practice, this means their APFs would simply store the element contents for later reference by other APFs. In the starter set, for example, the APFs for the title page and its elements are actually implemented in this way.

When an element is defined as possessing a certain attribute, a value for that attribute must always exist. Normally it is specified explicitly with an attribute label, but if a default value can be assumed when needed, your document type description can define that attribute's label as optional. In deciding whether an attribute label should be optional, you should balance such considerations as the savings in keyboarding time, the clarity of the markup with and without the label, and the ease with which your APFs can supply a default value.

The various techniques available in SCRIPT/VS for recognizing and processing attributes are discussed below in "Attribute Interpretation" on page 120.

DESCRIPTION DESIGN CONSIDERATIONS

Some considerations which affect the design of document type descriptions are discussed in this section. The discussions of those which affected the design of the starter set are illustrated with starter set examples.

NORMAL FORM AND OTHER FORMS

Document type descriptions should describe the "normal" form of a document -- that which is reflected in the published copies. When an individual document is marked up to conform to its type description, it should be adequately marked up for all processing. (This implies that processing programs should expect documents to be in the normal form.)

At times, though, you may need to process documents which are not in their normal form. For example:

- You may need to process incomplete and out-of-order drafts. (See "Interim Processing" on page 18.)
- You might need to convert from the normal form to an interchange standard agreed on with other users. This is discussed further under "Interchange Considerations" on page 127.
- You might want some of your markup editors and text entry operators to use special markup conventions that are more consistent with their previous training, or the characteristics of their keyboards.

You could try to reflect such needs in your document type descriptions, thereby defining the "normal" form to include aberrations. This would require all APFs to accommodate the variations, since an APF must be able to process any document which conforms to the type description. A better approach is to identify possible variations from the normal form in the specifications for the APFs, while leaving the document type description undistorted. In this way, only those APFs which actually must process the variant forms will be affected.

INFLUENCE OF EXPECTED PROCESSING

In theory, GML markup design should depend only upon the characteristics of the document. In practice, one has also to consider in a general way the kind of processing that can be anticipated. It can affect both your document type descriptions and the markup conventions you establish.

Effect on Document Type Descriptions

Consider a service manual which has elements that are "removal instructions" and other elements that are "replacement instructions." If you cannot reasonably anticipate a use of the document in which the two kinds of instructions would be processed differently from one another, it would simplify matters to consider them a single element type. Otherwise, you should play it safe and define them as separate element types. (You could of course associate their tags with a common APF for processing.)

In making such evaluations, you should consider the processing of the document by human beings as well as by SCRIPT/VS and other programs. In our example, if the authors and editors who create and revise the source document habitually treat "removal instructions" as something distinct from "replacement instructions," then that is a good enough reason for your markup to reflect the same distinction.

Processing restrictions can also affect markup design. For example, SCRIPT/VS terminates a keep if another keep begins within it.[23] Since the starter set APFs for figures and footnotes establish keeps, the descriptions of these elements prohibit them from occurring within the bounds of one another. Situations like this should be avoided, if possible, since they subject the permanent document data base to the limitations of the most restricted application. In the starter set, however, these particular restrictions were acceptable because they were not unnatural for the type of document. They might have been adopted even in the absence of the processing constraint.

Effect on Markup Conventions

The starter set includes a number of examples of the effect of anticipated processing on the design of markup conventions. The table of contents element, for one, has no content defined for it. This is because the supplied APF generates it automatically. For the same reason, no explicit ending tag is required for the element. If you define document types for which the table of contents will not be generated automatically, you should create a type description for the table of contents element.

The starter set APFs for figure and heading references generate not only the figure number or heading text, but the page number on which the element begins and some appropriate text as well. This saves entry keystrokes, but requires that the markup editor be aware of the generated text. If you should associate a different APF with the tag, it might generate different text which would no longer fit the context.

You may want to change these APFs, or not use these tags altogether. In the latter case, you can obtain similar function by using symbols, as described in "Reference to Generated Information" on page 126.

[23] This is a simplification, but is sufficiently accurate for this discussion.

RULES OR CONVENTIONS?

You can decide whether your document type descriptions will reflect rules, or merely conventions. An element APF, for example, might invoke only the attribute APFs for attributes validly specified for elements of that type, or it might do no such checking. Similarly, some APFs check to see if they are within the type of element in which their associated tags are permitted to occur, while others do not. When you specify your own APFs, you must balance the benefits of strict system-assisted enforcement of markup rules, against the economies of being able to use the same APF with a number of different tags, and the performance impact of extensive checking.

Strict enforcement of <u>syntax</u> rules, on the other hand, does not involve the same degree of negative side effects. When your APFs scan for attributes, they can -- and should -- check to see that delimiter characters were entered correctly and issue appropriate messages if they were not. The starter set includes some examples of how to do this, but does not provide complete enforcement.

FREEDOM OR COMPLEXITY?

By developing a document type description, you are able to understand exactly what tags can occur in a document, and where. When you then create a markup guide for that type of document, it could state clearly, for example, that the preface is optional, but if used it must occur after the abstract. This will guide the markup editors. Similarly, the text programmer will know that the preface APFs will always execute, if at all, after the abstract APFs. This could affect the way he programs the APFs.

At the lower levels of the document, where there can be considerably more variation in structure, this discipline becomes even more important. Your analysis will cause you to address -- and resolve -- questions which could otherwise puzzle markup editors and/or cause APFs to malfunction. For example:

Can lists occur within figures?

Can figures occur within lists?

Can lists occur within footnotes?

How you answer questions of this kind is not as important as the fact that you consider them and record your answers in your document type descriptions. You must balance freedom of document structure, on the one hand, against the potential complexities of implementing APFs that accommodate such freedom.

<u>SOME QUESTIONS TO ASK</u>

When creating document type descriptions, situations can arise which do not suggest a single correct way to handle them. Sometimes there really is no "correct" way; use whichever alternative you prefer. In other cases, though, you can improve your understanding of the situation by asking the kinds of question discussed in this section.

WHY IS THIS DONE?

Some formatting may appear random, like the italicization of phrases. By asking "Why is this done?" you can better determine the correct markup. The italicization might have been simply for emphasis, or it might have been a foreign word, or a bibliographic reference. If the differences among these three were important, you would define three different element types for precise identification. If not, a generic type like **hp1** (highlighted phrase) would do.

IS THIS TEMPORARY?

You should only use APFs for processing which is associated with the permanent characteristics of the document. This assures that your document will be usable for many purposes over a long period of time. Processing stemming from temporary characteristics, such as repairs to undesirable formatting results, is better handled with control words. (See "Control Words and Macros" on page 13 of the Markup Guide, and "Control Words" on page 127 in this Part.)

IS THE FORMATTED OUTPUT INVOLVED?

Persons experienced with formatting using specific control word markup frequently request a "new page" GML tag (or "new column," "new line," etc.). The possible reasons for the request are:

- To modify formatting results, such as to prevent a widow.

  This stems from temporary characteristics. A patch **psc** element and appropriate control words should be used.

- To cause an element to begin on a new page. For example, a preface might contain a second-level heading segment entitled "Acknowledgments" that the house style says should begin on a new page.

  This is not really a second-level heading segment, since the house style for second-level .heading segments does not require a page break. It is a unique element type which should have its own tag and APFs. The APFs would enforce the house style for acknowledgements by causing the page break.

- To create a special graphic effect, such as a three-column table.

  This should be defined as an element type, and APFs created, unless doing so would not be practical. For example, there may not be sufficient use of exactly this type of element, or the APF processing might of necessity be highly device-dependent. The alternative would be to treat the element as a graphic **psc** element and use control words to create the desired effect.

- To "copyfit" a document.

  Some documents, such as reference cards, must be written with the final layout in mind; the author must indicate page breaks in his source document. This is a case where the page is genuinely an element of the source document and should be identified by its own tag. However, it is best to call it a "panel" or "frame" or "display" to emphasize that it is a logical element, not just a physical page.

- The document contains many elements of the same type, each of which is printed on a separate page.

  For example, a service manual might have a single service procedure on each page (or pair of facing pages). Although there is a logical correspondence between elements of the source document and pages, the source elements (unlike those in the reference card example) can exist independently of the pages. A tag for "service procedure" should be defined, and APFs written that will cause the necessary page breaks.

This discussion raises an issue whose importance cannot be overemphasized: the distinction between processing instructions and descriptive tags. Disguising control words as tags to "keep formatting control words out of the document" is the same sort of error as using legitimate tags solely to invoke particular APFs. Neither practice is generalized markup, regardless of the delimiter being used. Both practices jeopardize the attainment of GML benefits.

IS PHYSICAL STORAGE INVOLVED?

A possible source of confusion in markup design can stem from failure to distinguish between the "logical" document you are marking up, and the "physical" representation in which it is stored.

For example, when you store a single document as a number of separate files, you do not ordinarily reflect this in the markup design. This is because the physical files need not have any correspondence to the logical structure. If you changed the number of separate files, the document would still look exactly the same to SCRIPT/VS, and the processing results would be the same. The combining of source files, in this instance, is not based on any characteristics of the document, and is best handled by the .IM (imbed) control word. (An analogy can be made to a typescript. Whether it is stapled in one group or many does not affect the comprehensibility of the document.)

In contrast, the conditional inclusion or exclusion of portions of the logical document (which may or may not be stored in separate files), is very much related to GML markup. This is discussed in the next section.

WHAT ABOUT MULTIPLE VERSIONS?

Your installation may produce a group of documents which differ from one another in only a few places. For example, they may be service manuals for a machine with slight differences based on the options purchased with it.

SCRIPT/VS offers powerful conditional processing capabilities for producing such multi-version documents with a minimum of duplication. GML can provide a very simple markup interface to these powerful capabilities.

For example, suppose there were three versions of a user's manual for office equipment to correspond to three different models that are available. The manuals are identical except for descriptions of operating procedures, since some models do different things automatically.

You could handle this situation in your document type description by defining an element type of "procedure" with an attribute of "model." In the source document, you would enter three versions of each procedure -- one for each model -- along with the common portions of the document.

When the document is processed, the profile (perhaps through a SYSVAR option) would indicate which model's manual was being produced. This would permit the "procedure" and "model" APFs to test the model attribute of the procedure against the model (i.e., version) of the document being processed. Versions of the procedure for the other two models would be ignored.[24]

DEFINING TAGS

When you analyze documents to define document type descriptions, you think in terms of the element type names and the names of the attributes. When you mark up documents, you use tags, which are actually mnemonic abbreviations for these names. (In this context, and for the remainder of this discussion, "tag" means both element type markup tags and attribute labels, exclusive of the delimiters.)

Choosing appropriate tags can be very important, and is subject to a variety of conflicting considerations. For example, short tags reduce data entry time and file size, while longer tags make it easier to remember what the tag stands for. In the starter set we used short tags for the most common elements, since frequent use would increase the likelihood of remembering them. You might consider resolving such conflicts by establishing long and short forms of tag for each attribute or element type. Both tags would always be associated with the same APF.

To conform to the GML syntax convention, tags should be one to eight characters long. The characters may be upper case A-Z and 0-9, with a leading alphabetic required. Blanks are not allowed in a tag. All of the tags in the starter set conform to this convention.[25]

---

[24] This is similar to what the **psc** and **proc** APFs do in the starter set.

[25] Since SCRIPT/VS actually interprets tags as symbols, it will accept lengths of up to 10 characters, and some special characters. This fact will not affect GML processing.

Although tags are treated internally as upper case, SCRIPT/VS permits you to adopt different practices regarding the entry of tags. In the starter set, we chose all lower case for ease of entry. You might prefer upper case because it distinguishes the markup more clearly from the text. (You can even mix cases within a tag, but this may make it more difficult to edit the document. Most text editing systems are case-sensitive; to locate a mixed case tag, you would need to remember exactly which letters were capitalized.)

In the starter set, when a tag is used to identify the end of an element, it is prefixed with an "E" (for "end"). This character is not included in the eight character limit for the tag proper. However, it is combined with the tag and the combination is entered in the SCRIPT/VS symbol table, just as "pure" tags are. Therefore, to avoid possible confusion, the starter set has no tags which begin with "E." You may choose to follow the same convention, or you might prefer to use the "#" special character instead of the "E." The special character offers the added benefit of distinguishing the ends of elements more clearly in the source document.

Since tags are treated as symbols, you must be sure there are no upper case symbols which are the same as tags (with or without the "E"). Lower case and mixed case symbols will not be confused with tags, since symbols (unlike tags) are sensitive to case.

Document type descriptions reflect both the natural characteristics of the documents and, to a lesser extent, the processing anticipated for them. This chapter explains the relationship between GML markup and SCRIPT/VS processing capabilities.

GML INTERPRETATION

GML documents are processed by:

1. recognizing the start or end of an element, or an attribute label;
2. associating it with an APF; and
3. executing the APF.

Recognition and association can be performed in a variety of ways, by both the SCRIPT/VS system and your own APFs. Moreover, recognition and association can be discrete steps, or be combined with one another, and/or with the execution of the APF. Each approach has important implications. This section discusses how these tasks can be performed for the start of an element, for attributes, and for the end of an element.

ELEMENT START INTERPRETATION

The start of an element can be marked up explicitly or recognized implicitly. In the explicit case, SCRIPT/VS recognizes the tag and associates it with an APF. An APF is implemented as a sequence of control words, possibly intermixed with text and symbols, in one of three forms: macro definition, value of a symbol, or imbedded file. The method of associating a tag and an APF depends on the form of the APF.

In all cases, SCRIPT/VS recognizes the GML delimiter and treats the element type tag as an upper case symbol. The value of the symbol, however, depends upon the form of the APF. If the APF is implemented as a macro, the value of the symbol will be an invocation of that macro, as in:

```
    .se TITLEP = '.@prolog '
```

Or the value could be the APF itself: a sequence of control words separated by control word separator characters (semi-colons in the starter set).

```
    .se TXMP = '.sk;.fo off;.in 3;.li on'
```

If you choose to implement the APF as a separate file, the symbol value will be an invocation of the imbed control word (.IM) with the APF name as the file to be imbedded.

```
    .se NOTICE = '.im notice78 '
```

It is sometimes useful to mix the forms of APF implementation. For example, some specialized elements are processed by generating text for a heading, and then processing that heading in the usual way. It is possible to create a separate macro for each such element, but this would be wasteful. The following example shows how such elements can share a common macro by including part of the APF in the symbol value and invoking macros for the remainder.

```
    .se REMOVAL = '.@h4 Removal Instructions;.@dl '
    .se ASSEMBLY = '.@h4 Assembly Instructions;.@dl '
```

An element can also be recognized implicitly by the tag for another element. In such a case, the recognition is performed by your APF, which also performs (or invokes) the processing for the other element. In the starter set, for example, a head one (**h1**) implies a first level heading segment. The value for the symbol could invoke two APFs, like this,

```
   .se H1 = '.hsegl ;.h1 '
```

Or (as actually implemented) it could invoke a single APF which performs the processing for both the segment (skipping to a new page) and the heading (changing font, and so on).

ATTRIBUTE INTERPRETATION

Attribute interpretation is completely under the control of the element start APF. It can ignore the possibility of attributes, or it can test for them (and take appropriate action) at any point in its processing. The element start APF could perform the attribute processing itself, but this would tend to make it complex and specialized, and restrict its usability for other applications and other element types. The starter set seeks to minimize this problem by defining an independent APF for each attribute.

An element type APF can test for attributes by using the starter set .@SCAN macro, and then execute them by invoking the .@EXATT macro. These macros map an attribute to an APF by treating the attribute tag as the name of a macro, and the attribute value as the macro's parameter list. For example, the markup

```
  :fig place='page'
```

results in the following attribute APF invocation:

```
   .place page
```

In this approach, the APF name must always be identical to the attribute label. Therefore, you can only change the APF mapping by having a number of macros with the same name and choosing among them. This is done by storing each of the synonymous macros in a different library (or a different profile), and specifying the desired library (or profile) with a SCRIPT/VS command option.

The macro definition for an attribute APF can be any of the things that the value of an element type tag symbol can be. In particular, it can be an invocation of another -- differently named -- macro that will perform the actual APF processing. Like tag symbols, macro names are converted internally to upper case, so you can use the same entry conventions for attribute labels that you do for tags.

You could perform attribute interpretation differently in your own installation by having your text programmer modify the .@SCAN macro. For example, you could store in a symbol in your profile an "index name." Your version of .@SCAN would prefix the index name to the attribute label to derive the APF macro name. For example, if the value of the index symbol were "ab," then our markup example would result in the following APF invocation:

```
   .abplace page
```

By defining new index names as needed, your APFs could have complete control over the mapping process.

Another possible variation of .@SCAN would be to treat the attribute label as a symbol. It would generate the following from our example:

```
  &place.page
```

This would provide the same capabilities for mapping attribute labels as exist for element type tags, although the symbol would be case-sensitive.

ELEMENT END INTERPRETATION

The end of an element can be recognized by SCRIPT/VS if it is explicitly marked, or it can be recognized implicitly by your own APFs. In the starter set, the explicit markup of an end of an element consists of ":E" and the element type tag. SCRIPT/VS recognizes the GML delimiter (:), and treats the "E" and the tag together as an upper case symbol. The association of the symbol with an APF is the same as described for the element start, for example:

```
   .se EOL = '.@el o'
```

Implicit recognition of the end of an element is simplest when the element is always followed by an element of a particular type, as in the **dt**, **dd** sequence in definition lists. Here, processing for the end of the **dt** can be invoked (or performed) when the **dd** is recognized, just as with implicit recognition of the start of an element.

However, when the structure of the document permits more variation, it may be necessary for earlier APFs to leave indications that will tell later APFs their location in the structure of the document. For example, the APF for the preface could set a symbol to indicate that the preface had begun. APFs for elements which could not occur within the preface would test that symbol. The first such APF to execute would therefore know that the preface had just ended and could perform the end of preface processing. It would then change the symbol value so no other APFs would perform end of preface processing by mistake.

End of element recognition is also implicit when the content of an element is "Text on Same Line," as with headings in the starter set. Here, the heading APF is able to "see" the entire content at once, and can thus perform the end of element processing as well as the processing at the start.

SPECIFYING APFS AND PROFILES

This section discusses how to organize profiles and APFs, and the ways in which optional processing can be specified.

WHAT IS A PROFILE?

A profile is a document that is automatically processed when SCRIPT/VS is invoked. Although, the .SE (set-symbol) statements that map tag symbols to APFs could be anywhere in a document, it is usually most convenient to collect them in the profile. The profile may also contain macro definitions and .SE statements that initialize variables used by the APFs to establish the formatting style. In the starter set, for example, the default indention for a simple list is set as follows:

```
    .se @sin = 4m
```

ORGANIZING YOUR APFS

A single APF could be capable of performing a variety of processing functions, chosen by the value assigned to certain symbols. Or, it could be a much simpler single-function routine.

In the former case, your tags would map to the same APFs in a variety of applications, but your profile would contain symbols that would be initialized differently for each application. The profile supplied with the starter set is set up this way. It results in fewer, but more complex, APFs. For example, heading numbering, number of columns, duplex printing, Others include the movement of periods and commas inside a quotation, are among the APF functions that are controlled by symbols in the starter set profile.

The alternative would be to have a greater number of simpler, single-purpose APFs. The tradeoffs to consider are available programmer skill levels, ease of managing your library of APFs, likelihood of changes to the APF specification, and ease of maintenance.

IMPLEMENTING PROCESSING OPTIONS

There are two classes of processing option:

- You may design your APFs so that by setting a symbol value you can get different types of processing that reflect different formatting styles. (Your establishment may have different house styles for different document types, for example.) These options should be specifiable only by the Document Administrator, not the Markup Editor. They are called "controlled options."

- You may want different processing for draft copies than you do for interim distribution or for final copies. You might want spelling checking for some runs, but not others, or you might want to initialize heading number counters for processing of part of the document. These are called "user options"

SCRIPT/VS offers the following techniques for specifying controlled options:

- Setting defaults in the permanent environment, such as logical device, profile name, etc.
- Setting symbol values within a profile.
- In the CMS environment, creating an OPTIONS file which will transparently invoke command options.
- Creating system procedures (CMS exec, JCL, procs, etc.) which will transparently invoke SCRIPT/VS with the desired command options.

User options can implemented in the following ways:

- Specifying a profile with the PROFILE command option.

  Even when, as with the starter set, a single profile handles all tag mappings for a document type, you might consider creating multiple profiles for each application. Each application profile would simply imbed the document type profile and set the variables and switches to achieve the desired processing. This will permit your Markup Editors to obtain desired application processing by specifying the appropriate profile when invoking SCRIPT/VS.

- Setting symbol values through the SYSVAR command option.
- Specifying a library of APFs and symbols with the SEARCH command option.

  The association of tags and APFs within a profile goes only to the point of identifying a symbol or macro name. SCRIPT/VS permits the same name to be associated dynamically with different definitions by specifying the library which is to be searched. This is done with a combination of command options and control words, depending upon the system environment.

- Specifying other command options, such as DEVICE, BIND, etc.
- Specifying an OPTIONS file (CMS only).
- Invoking SCRIPT/VS through a particular system procedure which includes the set of options needed for a particular application or processing run.

In addition to dividing options into the classes "controlled" and "user," it can also be useful to consider the distinction between those which affect GML interpretation (i.e., which APFs go with which tags) and those which alter APF parameters.

In general, GML interpretation is most effectively controlled by the "mapping" control words (.SE and .DM) within a profile, and by designating an active library. APF parameters are best varied by setting symbols within the profile, either directly, or by the user with SYSVAR, OPTIONS files, or system procedures. However, SCRIPT/VS offers such flexibility that you may find ways of intermixing these techniques, or developing others, to meet your own needs.

SUPPLEMENTING SCRIPT/VS PROCESSING

Your documents may require processing that SCRIPT/VS was not designed to perform. GML greatly facilitates using SCRIPT/VS in conjunction with other programs, as pre-processors, post-processors, or independent programs operating on the same source files.

GENERATING SCRIPT/VS INPUT

You can use SCRIPT/VS to format reports using data from data processing files. An application program could access these files, perform the necessary computations, and create an output file. This file could contain GML markup just as if it had been created with your usual text entry procedures. You will then be able to process it with the same flexibility as your other documents.

You can also use SCRIPT/VS to prepare input for itself. For example, SCRIPT/VS could help prepare an index to a manual. You might define an attribute called "index descriptor," make up a suitable attribute label ("desc"), and have your text programmer produce an APF. You would specify this attribute for those phrases in your document which were suitable entries for an index.

The APF would write the phrase and current page number to a file, together with appropriate element type tags identifying each. You could sort the file using a sort program and arrange the sorted file in a manner appropriate to an index. The sorted file would be a SCRIPT/VS document which, when processed by appropriate APFs, would generate an index to the manual.

USING SCRIPT/VS TO PREPARE INPUT FOR OTHER PROGRAMS

SCRIPT/VS has a great variety of general document handling functions which can be used independently of formatting. You can use these functions to create APFs which will translate a GML document into suitable input for another program, such as a formatter that can support photocomposers.

For example, the starter set APFs for ordered lists and list items automatically generate numbers (or letters) for the items on an ordered list. This is a desirable function, since it permits the list to be revised without renumbering all the items.

Those APFs include general processing control words which maintain the number counter, translate numbers to letters, and insert the numbers or letters into the output text stream. They also include formatting control words which indent the list and leave spaces between the items. You could create a modified version of the APFs which would retain the SCRIPT/VS controls that perform general processing functions, but would eliminate the SCRIPT/VS formatting controls. Instead, the APF would insert the appropriate control words of the postprocessor into the output stream.

In addition to creating APFs, you would also create a profile which would map the GML markup to the new APFs. The profile would also issue control words that would turn off justification and page numbering, and the like, so the output will look like a source file. You might also need to translate special characters which might be unacceptable to the post-processor.

By having two sets of APFs and two profiles, you could continue to proof the document on a line printer while getting final output on a photocomposer via the post-processor.

SHARING SCRIPT/VS GML FILES

SCRIPT/VS interprets GML markup by recognizing tags, associating them with APFs, and executing the APFs. You can write other programs which do the same thing, and the APFs could then be any function of which your programming system is capable.

If you create formal definitions of your document types, persons familiar with computer language processing techniques can write programs to verify the correctness of document markup, supply missing tags, or prompt markup editors in the creation of documents.

This chapter discusses considerations involved in defining markup conventions and procedures for your installation, and documenting them in GML markup guides.

GML MARKUP

GML syntax recognition is implemented using SCRIPT/VS symbol-processing facilities,[26] which tolerate the following deviations from the recommended syntax:

- The period (.) that separates markup from immediate text on the same line can be omitted if attributes are specified and there are no equal signs (=) in the text that follows the attributes.
- Quotation marks may be omitted for attribute values which do not contain blanks or special characters, but such values must be followed by a blank.

As a result, SCRIPT/VS may occasionally process incorrect GML markup as if it were marked correctly. If this is undesirable, you can develop your APFs to test for these conditions and issue appropriate warnings.

You can change the GML delimiter character by using the .DC GML control word, but you cannot change the period that separates markup from text. If necessary to meet special requirements of your installation, you can change the equal sign (=) which delimits attribute labels, and the single quotation marks that delimit attribute values, by modifying the starter set .@SCAN macro.

TEXT ENTRY

You should specify the desired and maximum line lengths based on the types of device your text entry operators and editors will be using, bearing in mind that expansion of symbols will lengthen many lines. You should also mention any prohibited keys that are on these devices, such as the "index return" on magnetic card typewriters.

You may also wish to redefine certain keys on the input keyboards used in your installation. This can be done with combinations of the translate-input (.TI) and translate-output (.TR) control words.

Or you could establish implicit markup conventions -- for example, by defining a leading blank to be equivalent to a **p** tag. This capability is provided by the SCRIPT/VS macro facilities.

SCRIPT/VS SYMBOLS

In addition to the uses mentioned in the Markup Guide, symbols are used extensively in implementing macros and APFs. Such symbols are normally not visible in the source document except where processing controls are entered. This discussion will therefore focus entirely on symbols used in conjunction with GML markup.

Note: Remember that GML tags are treated as uppercase symbols. You must not define uppercase symbols which are identical to GML tags, even if you enter tags in lower case.

[26] But GML tags are not processed identically to symbols, and GML and symbols have distinctly different functions in document markup.

ENTERING UNKEYABLE CHARACTERS

The choice of symbols for this purpose will depend upon the types of keyboard and printer in use in your installation, and upon the types of document processed. Each of your markup guides might therefore define a different set of special character symbols. Figure 17 suggests some candidates, based upon the character set of the IBM TN print train.

&bul.	Bullet	&sup9.	Superscript 9
&lbr.	Left bracket	&supr.	Superscript ")"
&rbr.	Right bracket	&supl.	Superscript "("
&lbrc.	Left brace	&suppl.	Superscript plus
&rbrc.	Right brace	&supmin.	Superscript minus
&ltsym.	Less than	&deg.	Degree
&lesym.	Less or equal	&sqbul.	Square bullet
&nesym.	Not equal	&loz.	Lozenge
&gesym.	Greater or equal	&bxul.	Box upper left
&gtsym.	Greater than	&bxur.	Box upper right
&notsym.	Not sign	&bxll.	Box lower left
&plusmin.	Plus or minus	&bxlr.	Box lower right
&sup0.	Superscript 0	&dash.	Dash

Figure 17. Possible Symbols for Unkeyable Characters

The symbols would be set to the printer codes for the desired characters. The definition of the "&rbl" symbol in the starter set profile is an example of how to define symbols for unkeyable characters.

You can use the conditional processing facilities of SCRIPT/VS to vary the definition of these (or any other) symbols. For example, if you define a symbol for a vertical rule, you might want it to have a different value for the 3800 than you do for line printers.

REFERENCE TO GENERATED INFORMATION

Symbols can let you refer to information which will not exist until your document is processed. For example, there are system symbols which let you refer to the current date, or the page number. You may wish to document these symbols for use by your markup editors.

Symbols can also permit forward and backward cross-references. As an example, the starter set APF for the ID attribute creates a symbol composed of the "@" special character, followed by the first two characters of the element name (with the first capitalized), followed by the value of the ID (as entered), followed by "pg." Thus, for the markup

```
   :fig id='ajs'
```

the APF generates the symbol "@Fiajspg." The value of the symbol is the page number on which the element is printed. You can define similar symbols in your own APFs, and document them for direct use in markup. Or, as we did in the starter set, you can reserve use of the symbols to APFs associated with "reference" elements.

ABBREVIATING LENGTHY PHRASES

In certain of your document types, some lengthy phrases (such as product or department names) may occur frequently in the text of the document. In such cases you may wish to define a symbol to be used as an abbreviation for the phrase. (When such phrases appear in connection with specific document elements, such as headings in a form, they can be generated by the APFs, and no symbols will be required.)

CONTROL WORDS

Situations will occur in which it will be necessary to supplement GML markup with control words. You should strive to create conventions for control word markup which will not prevent access to all of the functions of SCRIPT/VS (and any programs for which you may be using it as a pre-processor).

At the same time, you will wish to preserve the generality of your documents so they can be used for other applications, and so they can be processed by other programs you may use in conjunction with SCRIPT/VS. This means you should have some way for both people and programs to easily determine when and why control words have been used in the markup. The Markup Guide does this by identifying four categories of control word use:

- Source document management.
- Permanent formatting graphic effects (graphic **psc**).
- Temporary formatting repairs (patch **psc**).
- Non-formatting ("preprocessor") functions (profile **psc**).

The categories represent two approaches to the problem: unrestricted use of a limited set of control words, and carefully bounded use of an unlimited (i.e., full) set of control words. An implication of this organization of control words is that all of your markup editors can (and should) be taught the limited set. When the other situations for control words occur, the document should be given to a specially-trained person for markup of that portion.

The "limited set" control words are of an application-independent nature themselves, and will not harm the generality of your document. However, SCRIPT/VS macro facilities will permit you to redefine them if necessary (for example, to be comments which are ignored in processing, or to generate markup for a post-processor). You may wish to make other control words of this type (such as .SV for spelling verification and .HW for hyphenating a word at designated points), or macros of your own devising, available to your markup editors.

If you adopt the **psc** element conventions for bounding the use of the full set of control words, you might wish to have separate tags for patch, graphic, and profile, while continuing to map all three to the same APF for most processing. This will simplify markup, by eliminating the need for the separate comment line which identifies the type of control word use. Moreover, in applications which require differing treatment (such as ignoring all patches when preparing a document for interchange), you will be able to associate them with different APFs.

You can simplify the entry of short **psc** elements by defining macros, such as the following:

```
.PATCH 'procnames' .cw parameters
.GRAPHIC 'procnames' .cw parameters
.PROFILE 'procnames' .cw parameters
```

The macros could either perform the function of the **psc** and **proc** APFs, or they could generate the normal form of the document if required for interchange.

INTERCHANGE CONSIDERATIONS

One of the benefits of GML markup is that it can facilitate the exchange of documents among different users with possibly different processing requirements. This section discusses considerations in establishing interchange standards, and in preparing documents for interchange.

ESTABLISHING INTERCHANGE STANDARDS

An interchange standard could range in detail from an agreement to use GML markup syntax only, to a full interchange of documents, APFs, profiles, and libraries. The following list is ordered by increasing degrees of interdependence:

1. All parties use GML syntax and markup conventions. There is no agreement on standard document types or sets of tags.

2. An interchange document type is defined and documented by a document type description which describes its normal form.

3. A number of document types are defined and documented.

4. Standard APFs are developed for all processors (not just SCRIPT/VS) in use by the parties to the interchange standard. Interchange documents need not conform to standard document types, but must be accompanied by profiles which map tags to the standard APFs.

5. All parties use SCRIPT/VS and interchange documents, APFs, profiles, and libraries.

The choice of interchange standards depends upon how closely the parties to the interchange work together, how closely their processing environments match, and the ways in which they use one another's documents.

PREPARING DOCUMENTS FOR INTERCHANGE

The normal form of your documents may differ from the interchange standard. Your entry convention may differ from both of these (as discussed in "Normal Form and Other Forms" on page 113). Many of the considerations and techniques are the same in converting a local standard to an interchange standard, as in converting from an entry convention to a local standard.

Depending upon the level of standardardization and level of interdependence, as discussed above, you may need to:

- Convert your GML delimiter if it differs from the standard. If the standard permits varying delimiters, make sure that a control word or macro which defines the delimiter precedes any tags in your document.
- Replace any local symbols by their values (when possible).
- Translate characters to the standard character set.
- Resolve file imbeds.
- Remove unnecessary control words, such as patch and graphic **psc** elements for processes which the receiving party will not perform.
- Remove references to the local environment, such as local file names.

As an example, consider converting a specialized document type such as a GML Markup Guide to a more general interchange standard, such as the starter set General Document. You would use SCRIPT/VS as a preprocessor, as discussed in "Generating SCRIPT/VS Input" on page 123. Your text programmer would prepare APFs for each of the custom tags which would generate the appropriate General Document tags, together with any boilerplate text and **psc** elements.

Figure 12 on page 107, and Figure 13 on page 108, respectively illustrate a typical source and result of such processing. The conversion APFs also resolved symbols used as a convenience in entering lengthy or complex markup sequences, and performed other applicable functions as discussed above.

CREATING YOUR OWN GML MARKUP GUIDES

Throughout Part Three we have been discussing the design of your own GML support, using the starter set as a point of departure. Since the Markup Guide, which we have also been using as an example, describes the starter set document type[27] -- General Document -- it follows that it can be used as a point of departure for developing documentation for your own document type descriptions.

[27] The Markup Guide describes suggested use of the starter set. Examination of the macros and profiles themselves will reveal additional APF functions.

COMMUNICATION CONSIDERATIONS

Some distinctions which are important to GML markup design are of less importance to the person actually marking up documents. In the starter set Markup Guide we have taken a number of steps to simplify the information presented to the markup editor without introducing technical inaccuracies.

- The term "attribute" is used only to refer to an element's secondary characteristics. There is no mention of the fact that type and content are also characteristics -- hence, attributes -- of an element.
- Attribute tags are always distinguished from element type tags by referring to them as "attribute labels."
- The terms "tag" and "label" frequently are used in a way that encompasses the delimiters as well as the tag (or label) proper.
- Attributes are characterized as "optional" or "mandatory," although these terms properly apply only to the attribute labels.

DESCRIBING PROCESSING

Markup guides should carefully distinguish descriptions of an element's characteristics from descriptions of processing performed by the APFs. This can be difficult in cases where the document type description has been influenced by anticipated processing, as when the content of the element is generated by an APF. However, it is relatively easy to refrain from gross errors (for example, describing the security level attribute as a "running heading").

In each tag description, we reserved a separate heading for discussing processing, and identified the profile whose processing was being described. Additional discussions could be added for other profiles that you may develop in your installation.

You can assist markup editors experienced with processing-oriented markup to adapt to descriptive markup by the way you prepare your markup guides. You might even consider removing the processing discussions entirely, and documenting the APFs elsewhere for reference by yourself and the text programmer.

DETERMINING YOUR REQUIREMENTS

Your requirement for GML Markup Guides may vary from a single guide documenting all tags and processing provided by your installation, to an individual guide for each document type. In making your decision regarding your requirements, the following factors may be significant:

- The number of document types.
- The extent of overlap of tags among them. (Are they all mostly common, or quite distinct?)
- The number of different markup editors, and the number of different document types each one handles. A wide variety would suggest a single guide, with distinctions between types called out; concentration on a single type would suggest a specialized guide, to avoid confusing the markup editor with information about irrelevant documents.

You might also require an operations manual to provide instruction on entering and revising files, managing files, and invoking SCRIPT/VS for each system environment your installation supports. This information could be included in each markup guide; or, if a single markup editor would typically use many guides, it might be less confusing to provide information which cuts across all document types in a separate manual. Some information on these subjects is provided in Part Two of this manual. This information is also distributed in machine-readable form to provide a base for your own customized documentation, subject to the procedures described in "Using the Starter Set Markup Guide" on page 130.

GML SUPPORT FOR MARKUP GUIDES

The tag description section of the Markup Guide uses a number of specialized tags and symbols. The APFs and symbol definitions are contained in a special profile included on the product distribution tape. Some features of the APFs are discussed in "Determining Document and Element Types" on page 106. Further understanding of their use, and the proper markup, can be had from examining the profile, the machine-readable copy of the starter set Markup Guide, and "Appendix F: Document Type Description for GML Markup Guides" on page 148.

USING THE STARTER SET MARKUP GUIDE

Parts One and Two of this manual, and Appendixes A, B, C, and D, have been made available with the Document Composition Facility licensed program in a machine-readable ("soft copy") form, for use both as a sample problem and as a base for the development of customized GML markup guides and other support documentation for use in a SCRIPT/VS installation. The Program Directory distributed with the Document Composition Facility contains information on how this file is distributed.

These materials are copyrighted by the IBM Corporation and contain the following copyright information:

> (C) Copyright International Business Machines Corporation 1978.
>
> This <u>GML Markup and Processing Guide</u> is copyrighted and is provided in machine-readable form to facilitate customer customization of the GML interface to the Document Composition Facility licensed program.
>
> Permission is granted to:
>
> - Store the copyrighted material in your system incorporating the Document Composition Facility licensed program and display it on that system's terminals. This permission is granted for the duration of the licensed program license.
> - Print only the number of copies required for use by those persons who are using the Document Composition Facility licensed program on the system for which this material is provided.
> - Modify the material to meet specific installation requirements.
>
> The IBM copyright notice must appear on all printed versions of this material or extracts thereof. Permission is not granted to further reproduce or distribute the material.

MARKUP PROCEDURES

ENTERING GML MARKUP AND TEXT

The rules for GML markup are:

1. Document elements are identified with GML tags. The tags begin with the GML delimiter, which is defined in the starter set profile as a colon (:). Begin a new input line for all tags except text items.

2. For some elements, additional characteristics can be specified with attribute labels. Attribute values must be enclosed in single quotation marks. For example:

   ```
   :fig depth='7i'
   :efig
   ```

   Recommended space units are inches (nnI), millimeters (nnW), picas/points (nnPnn), or Ciceros/Didot points (nnCnn).

3. When the content of an element is immediate text, it must begin on the same line as the markup.

4. When text follows markup on the same line, enter a period (.) before the text.

5. Start all input lines at the left margin.

6. When you end a sentence, do not enter any more text on that line.

7. Do not start text lines with a period.

The following practices are also recommended:

1. Do not mark an element with a tag which does not correctly describe it, even if it results in the correct processing.

2. Enter tags and attribute labels in lower case.

3. Do not enter blank lines.

4. Maximum input line length is 132 character positions. (About 70 is recommended, for ease of revision.)

5. Do not underscore or overstrike characters.

6. Do not use the tab key.

7. When storing parts of a document in separate files, be sure each file starts at the beginning of a paragraph unit or higher level element, and ends at the end of one.

SCRIPT/VS SYMBOLS

Symbols begin with an ampersand (&) and end with a period (.).

&amp.	For printing the & character without treating it as a symbol delimiter.
&gml.	For printing the current GML delimiter (: in the starter set).
&rbl.	For entering a required blank character.

Source Document Management

Use these control words and macros freely within a document:

.im fileid	Includes stored file named by fileid in your document.
.cm text	Text is a comment in your source file.
.rc	Consult document administrator for use of revision codes.
.att label='val'	Inserts attribute labels and values into following line.

PSC Elements

Use PSC elements to enter control words and macros for the following purposes:

- Graphic Formatting

```
    :psc proc='procname'
    .cm graphic
    (specific control words)
    :epsc
```

- Modifying Processing Results

```
    :psc proc='procname'
    .cm patch
    (specific control words)
    :epsc
```

- Additional General Processing

```
    :psc proc='procname'
    .cm profile
    (specific control words)
    :epsc
```

Valid process names are the logical and physical device names in Figure 19 on page 140.

TAGS FOR GENERAL DOCUMENTS

The starter set is intended for use with documents whose type is "General Document." A wide variety of documents can conform to this type. The structure and elements of a General Document are shown in Figure 18 on page 133. The tables following the figure show the usage rules for the tags, and the permissible values for the attribute labels.

```
•   Overall Structure:
        General Document (GDOC)
            front matter (FRONTM)
                title page (TITLEP)
                    document title (TITLE)
                    document number (DOCNUM)
                    document date (DATE)
                    author name (AUTHOR)
                    address of author or publisher (ADDRESS)
                        address line (ALINE)
                abstract (ABSTRACT)
                preface (PREFACE)
                table of contents (TOC)
                list of illustrations (FIGLIST)
            body (BODY)
                part (implied by H0) or chapter (implied by H1)
            appendix section (APPENDIX)
                first-level heading segment (implied by H1)
            back matter (BACKM)
                first-level heading segment (implied by H1)

•   Heading Segments:
        nth-level heading segment (implied by H0 to H6)
            head n (H0, H1, H2, H3, H4, H5, H6)
            basic document elements
            lower-level heading segments

•   Basic Document Elements:

    -   Paragraph Units:
            note (NOTE), paragraph (P), paragraph continuation (PC)

    -   Lists:
            definition list (DL)
                definition term (DT)
                definition description (DD)
                list part (LP)
            ordered list (OL), simple list (SL), unordered list (UL)
                list item (LI)
                list part (LP)

    -   Other Basic Document Elements:
            address (ADDRESS)
                address line (ALINE)
            example (XMP)
            figure (FIG)
                figure body (no tag)
                figure caption (FIGCAP)
                figure description (FIGDESC)
            long quotation (LQ)

•   Text Items:
        figure reference (FIGREF)
        footnote reference (FNREF)
        heading reference (HDREF)
        highlighted phrase (HP0, HP1, HP2, HP3)
        phrase (PH)
        quote (Q)

•   Special Elements:
        footnote (FN)
        process-specific controls (PSC)
```

Figure 18. Summary of General Document Tags

Type	Attributes	Content	Termination
:abstract	None	No Immediate Text	:preface, :toc, :figlist, or :body
:address	None	No Immediate Text	:eaddress
:aline	None	Text on Same Line (no other tags)	At End of Line
:appendix	None	No Immediate Text	:backm or :egdoc
:author	None	Text on Same Line (no other tags)	At End of Line
:backm	None	No Immediate Text	:egdoc
:body	None	No Immediate Text	:appendix, :backm, or :egdoc
:date	None	Text on Same Line (no other tags)	At End of Line
:dd	None	Implied P Structure	:dt, :lp, or :edl
:dl	termhi= tsize=	No Immediate Text	:edl
:docnum	None	Text on Same Line (no other tags)	At End of Line
:dt	None	Text on Same Line (no other tags)	At End of Line
:fig	id= depth= place= frame=	No Immediate Text	:efig
:figcap	None	Text on Same Line (no other tags)	At End of Line
:figdesc	None	Implied P Structure	:efig
:figlist	None	Generated by APF	At End of Markup
:figref	refid=	Generated by APF	At End of Markup
:fn	id=	Implied P Structure	:efn
:fnref	refid=	Generated by APF	At End of Markup
:frontm	None	No Immediate Text	:body
:gdoc	sec=	No Immediate Text	:egdoc
:hdref	refid=	Generated by APF	At End of Markup
:hp0	None	Text Item	:ehp0
:hp1	None	Text Item	:ehp1
:hp2	None	Text Item	:ehp2
:hp3	None	Text Item	:ehp3
:h0	id= stitle=	Text on Same Line (no other tags)	At End of Line See "Usage" in text
:h1	id= stitle=	Text on Same Line (no other tags)	At End of Line See "Usage" in text
:h2	id=	Text on Same Line (no other tags)	At End of Line See "Usage" in text

Type	Attributes	Content	Termination
:h3	id=	Text on Same Line (no other tags)	At End of Line See "Usage" in text
:h4	id=	Text on Same Line (no other tags)	At End of Line See "Usage" in text
:h5	None	Text on Same Line (no other tags)	At End of Line See "Usage" in text
:h6	None	Text on Same Line (no other tags)	At End of Line See "Usage" in text
:li	None	Implied P Structure	:li, :lp, :eol, :esl, or :eul
:lp	None	Implied P Structure	:dt, :li, :edl, :eol, :esl, or :eul
:lq	None	No Immediate Text	:elq
:note	None	Paragraph Unit	Paragraph or Higher Level Element
:ol	None	No Immediate Text	:eol
:p	None	Paragraph Unit	Paragraph or Higher Level Element
:pc	None	Paragraph Unit	Paragraph or Higher Level Element
:ph	hi=	Text Item	:eph
:preface	None	No Immediate Text	:toc, :figlist, or :body
:psc	proc=	See "Usage" in text	:epsc
:q	None	Text Item	:eq
:sl	None	No Immediate Text	:esl
:title	stitle=	Text on Same Line (no other tags)	At End of Line
:titlep	None	No Immediate Text	:etitlep
:toc	None	Generated by APF	At End of Markup
:ul	None	No Immediate Text	:eul
:xmp	depth=	No Immediate Text	:exmp

Permitted attribute values are described in the following list. Enclose attribute values in single quotation marks. Attributes are optional unless otherwise stated.

depth=	Reserved vertical space in space units.
frame=	Figure emphasis: 'rule' or 'box' or 'none' or another character string.
hi=	Highlight type 0, 1, 2, or 3.
id=	One to five alphameric characters unique among elements referred to by the same type of reference element. (Mandatory for **fn** elements.)
place=	Figure placement: 'column' or 'inline' or 'page'.

proc=	One or more names, separated by blanks, that identify process(es) for **psc** elements. Each name is one to eight upper case alphameric characters.
refid=	One to five alphameric characters that are the **id** attribute of the element referred to. (Mandatory attribute.)
sec=	Character string identifying the security classification.
stitle=	Character string that is a short version of a heading ortitle.
termhi=	Highlight type 0, 1, 2, or 3 for definition terms.
tsize=	Maximum number of characters in a definition term, plus one (entered 'nnM').

PROCESSING GENERAL DOCUMENTS

PROFILE AND VARIATIONS

Profile(s) usable with General Documents are:

gdocprof Produces formatted output on terminals, line printers, and the IBM 3800 Printing Subsystem. Verifies spelling when "SPELLCHK" option is specified.

SYSVAR parameters for variations are:

c 1, c 2	Columns: Prints number of columns specified (1 or 2). If omitted, prints 2 on the 3800, and 1 on other devices.
d yes	Duplex: Begins new odd-numbered page for **h0**, **h1**, and front matter segments. If omitted, begins next available page.
h num	Head Numbering: Automatically numbers heading types **h1**, **h2**, **h3**, and **h4**. If omitted, prints without numbering them.
p process-name	Process: Specifies process for **psc** elements. If omitted, the names of the physical and logical devices are used.
t no	Title Page Printing: Suppresses printing of the title page.

INTERIM PROCESSING

The GDOCPROF profile permits these shortcuts:

- **toc** and **figlist** can be processed as last two tags of back matter.
- Heading segments and other elements can be processed independently.

ERRORS

Common causes of errors are:

- Omitting the period at the end of a symbol.
- Omitting the period at the end of markup (when required).
- Omitting the ending tag of a list, figure, example, or other element that requires explicit termination.

- Ending a quote (**q**) with a quotation mark instead of **eq**.
- Using tags where not permitted, such as within an element whose content is "Text on Same Line."
- Not entering a text item terminating tag immediately after the element.
- Omitting the tag for a paragraph that requires explicit markup.

SCRIPT COMMAND

The first parameter of the command is an identifier of the file or data set which contains the beginning (or all) of your document. For example:

```
  SCRIPT  mydoc
```

The file name may be followed by one or more options. (In CMS, a left parenthesis precedes the options.)

OPTIONS

The following list summarizes those options most likely to be used by markup editors. (Your installation may require other options as well.) The upper case letters signify acceptable abbreviations for the options.

Processing Options:

PROfile(name)	Name of file or data set to be used as document profile.
SYsvar(x value)	One or more pairs of variation codes ('x') and corresponding values.
SPellchk	Verifies spelling when requested with the .SV control word.
TWopass	Processes input document twice, to resolve forward references.
Bind(n1 n2)	Shifts printed portion of output page to right n1 space units on odd pages, and n2 space units on even pages, to leave room for binding. If n2 is omitted, n1 is used for all pages.

Logical Device and Output Destination:

DEVice(type)	Specifies logical device, and underlying physical device (which is also output destination). (See Figure 19 on page 140.) If omitted, "term" is the default in CMS and TSO, and "1403W6" in batch.
File(name)	Formats output for logical device, but stores in named file.
PRInt	Formats output for logical device, but prints on system printer. If no logical device is specified, "device(1403W6)" is assumed.
TErm	Formats output for logical device, but displays on terminal. If no logical device is specified, "device(term)" is assumed. (CMS and TSO only.)

Error Handling:

Message(delay)	Prints error messages at end of output document. (CMS and TSO only.)
COntinue	Continues processing after non-severe errors.

FILE NAME QUALIFIERS

In CMS, a default filetype of "script" is assumed.

In TSO, data set names assume "userid." and ".text" as default qualifiers.

LOGICAL	PHYSICAL	LPI	WIDTH	LENGTH
1403N6	1403	6	8.5	11
1403W6	1403	6	14	11
1403SW	1403	6	8.5	11
1403N8	1403	8	8.5	11
1403W8	1403	8	14	11
3800N6	3800	6	8.5	11
3800W6	3800	6	14	11
3800N6S	3800	6	11	8.5
3800W6S	3800	6	14	8.5
3800N8	3800	8	8.5	11
3800W8	3800	8	14	11
3800N8S	3800	8	11	8.5
3800W8S	3800	8	14	8.5
3800W12	3800	12	14	11
3800W12S	3800	12	14	8.5
TERM	2741	6	14	11

Figure 19. Device Names Supported by the Starter Set: Physical device name 1403 includes 3211; 2741 includes all terminals. Dimensions are lines per inch and default page size.

The following messages are generated by the starter set APFs. In all messages, the phrase "on page 'nnn'" refers to the page of the output on which the error can be found.

These messages are always preceded by three plus signs (+++) when displayed or printed. Messages which do not begin with three plus signs are SCRIPT/VS messages. Consult your document administrator about any SCRIPT/VS messages you receive.

DSMGML201W **The ID 'xxx' on 'yyy' on page 'nnn' used before**

<u>Explanation</u>: The document element whose type is 'yyy' has the same id='xxx' attribute as one or more other elements of the same type.

DSMGML203W **'xxx' tag on page 'nnn' implicitly ended list(s)**

<u>Explanation</u>: The tag 'xxx' was incorrectly placed within the bounds of a list. The list, and any lists nested within it, were terminated.

DSMGML204W **Redundant list end was found and ignored on page 'nnn'**

<u>Explanation</u>: An **edl**, **eol**, **esl**, or **eul** tag was incorrectly placed outside the bounds of a list.

DSMGML205W **No definition term for description 'xxx' on page 'nnn'**

<u>Explanation</u>: A **dd** tag was encountered without its preceding **dt** tag.

DSMGML206W **Definition 'xxx' on page 'nnn' but no defn. list start**

<u>Explanation</u>: The 'xxx' element is a definition list element (**dt** or **dd**) which was incorrectly placed outside the bounds of a definition list.

DSMGML207W **List was ended by 'xxx' list tag on page 'nnn'**

<u>Explanation</u>: The 'xxx' list element ending tag (**edl**, **eol**, **esl**, or **eul**) was erroneously used to end a list element of another type.

DSMGML208W **ID 'xxx' on page 'nnn' is too long, truncated to 'yyy'**

<u>Explanation</u>: The ID value 'xxx' exceeded 5 characters in length. It was truncated to 'yyy'. (If another element of the same type also has the ID value 'yyy', the unexpected duplication could cause message DSMGML201W to be issued for that element.)

DSMGML210S **'xxx' type lists not supported**

<u>Explanation</u>: There is an error in your document profile. The profile is attempting to invoke the list APF for a nonexistent list type. Consult your document administrator.

DSMGML211W **Extra end of highlight ignored on page 'nnn'**

<u>Explanation</u>: An explicit tag for ending a highlighted phrase (**ehp0**, **ehp1**, **ehp2**, or **ehp3**) was encountered without a preceding highlighted phrase tag (**hp0**, **hp1**, **hp2**, or **hp3**).

DSMGML212W **'xxx' found outside of 'yyy' on page 'nnn'**

<u>Explanation</u>: The 'yyy' element is a figure (**fig**) or an example (**xmp**). The 'xxx' element is either a figure caption (**figcap**), figure description (**figdesc**), or end of figure (**efig**) that was found outside of a figure, or an end of example (**exmp**) that was found outside of an example.

DSMGML213W **Invalid syntax for 'xxx' attribute 'yyy' on page 'nnn'**

<u>Explanation</u>: The attribute label 'xxx' and/or its value 'yyy' were entered incorrectly.

The following page is the title page created by the markup examples in the tag descriptions, such as the example in "titlep -- title page" on page 78.

GML Markup and Processing Guide

Document Number

MG-00001

July 3, 1978

L. T. Smith, Document Administrator

Any Company
500 Main Street
Anycity, Anyplace

Internal Use Only

APPENDIX E: DOCUMENT TYPE DESCRIPTION FOR GENERAL DOCUMENTS

The General Document description can apply to a variety of documents, from memoranda to technical manuals. It may be used as a "catch-all" category for documents which do not conform to any other type description.

ELEMENT TYPES AND STRUCTURE DEFINITION

This information is presented formally[28] in Figure 20 on page 147. The definition uses the tokens "sentence," "space," "word," "list," and "textitem," as well as those on the following list:

bde	Basic Document Element
char	Character
figbody	Figure Body
hseg	Heading Segment
ip	Implied Paragraph
line	Line Element
part	Zero Level Heading Segment
pcs	Paragraph Continuation Structure

The following notation is used:

:	Explicit Tag
=	Attribute
.	Content
,	Sequence
?	Optional
*	Repeatable
/	Alternative
( )	Grouping
[ ]	Restriction

PERMITTED ATTRIBUTE VALUES

depth=	Reserved vertical space in space units: inches (nnI), millimeters (nnW), picas/points (nnPnn), or Ciceros/Didot points (nnCnn).
frame=	Figure emphasis: 'rule' or 'box' or 'none' or another character string.
hi=	Highlight type 0, 1, 2, or 3.
id=	One to five alphameric characters unique among elements referred to by the same type of reference element.
place=	Figure placement: 'column' or 'inline' or 'page'.
proc=	One or more names, separated by blanks, that identify process(es) for **psc** elements. Each name is one to eight upper case alphameric characters.
refid=	One to five alphameric characters that are the **id** attribute of the element referred to.
sec=	Character string identifying the security classification.
stitle=	Character string that is a short version of a heading ortitle.
termhi=	Highlight type 0, 1, 2, or 3 for definition terms.

[28] Persons familiar with formal grammars will recognize that interword separators and sentence terminating sequences are ignored, and that the terminal characters ("char") are not defined. These omissions do not affect the usability of the definition for markup design.

tsize=	Maximum number of characters in a definition term, plus one (entered 'nnM').

MARKUP CONVENTIONS

These are the conventions for symbols and control word usage.

SYMBOLS

&amp.	& character -- not treated as symbol delimiter.
&gml.	GML delimiter character -- not treated as markup.
&rbl.	Required blank character.

CONTROL WORDS AND MACROS

Freely usable: .IM, .CM, .RC, .ATT

psc Elements: graphic, patch, profile

```
Overall Structure:
1.  :gdoc sec=?.frontm, body, appendix?, backm? :e
2.  :frontm.titlep, abstract?, preface?, toc?, figlist?
3.  :titlep.title, docnum?, date, author?*, address?* :e
4.  :address.aline* :e
5.  :abstract :preface.bde?*, hseg6?*, hseg5?*, hseg4?*, hseg3?*, :hseg2?*
6.  :body.part*/hseg1*
7.  :appendix :backm.hseg1*

Heading Segments:
8.  part.h0, bde?*, hseg1*
9.  hseg1.h1, bde?*, hseg6?*, hseg5?*, hseg4?*, hseg3?*, hseg2?*
10. hseg2.h2, bde?*, hseg6?*, hseg5?*, hseg4?*, hseg3?*
11. hseg3.h3, bde?*, hseg6?*, hseg5?*, hseg4?*
12. hseg4.h4, bde?*, hseg6?*, hseg5?*
13. hseg5.h5, bde?*, hseg6?*
14. hseg6.h6, bde?*

Basic Document Elements:
15. bde.list/address/fig/lq/note/p/pcs/xmp
16. list.dl/ol/sl/ul
17. :dl termhi=? tsize=?.lp?, ((dt, dd), lp?)* :e
18. :ol :sl :ul.lp?, (li, lp?)* :e
19. pcs.(ip/p/note), ((address/list/lq/xmp/fig), pc)*
20. :dd :li.ip, bde?* [no fn]
21. :lp.ip, bde?* [no fn, list]
22. :lq.bde?* :e [no fig]
23. :xmp depth=?.space/(line/bde)?* :e [no fig, fn, xmp]
24. :fig id=? depth=? frame=? place=?.figbody, (figcap, figdesc?)? :e
25. figbody.space/(line/bde)?* :e [no fig, fn]
26. :figdesc.ip, bde?* [no fig, fn]

Paragraph Units:
27. ip :note :p :pc.sentence?* [no fn]
28. sentence.(word/textitem)*
29. textitem.figref/fnref/hdref/hp0/hp1/hp2/hp3/ph/q
30. :hp0 :hp1 :hp2 :hp3 :q.(word/textitem)* :e
31. :ph hi=?.(word/textitem)* :e [no ph]
32. word.char*

Generated Elements (terminated by end of markup):
33. :figref refid=? :fnref refid=? :hdref refid=?.Generated by APF
34. :figlist space :toc.Generated by APF

Text Lines (terminated by end of input line):
35. :docnum :date :author :aline :h5 :h6 :dt :figcap.word?* [no psc]
36. :title stitle=? :h0-:h1 id=? stitle=? :h2-:h4 id=?.word?* [no psc]
37. line.(word/textitem)*

Special Elements:

Footnotes can occur wherever a bde can occur unless restricted.
38. :fn id=.ip, bde?* :e [no fig, fn]

Process-specific controls can occur wherever not restricted.
39. :psc proc=?.anything readable by SCRIPT/VS :e [no psc]
```

Figure 20. Formal Definition of a General Document

APPENDIX F: DOCUMENT TYPE DESCRIPTION FOR GML MARKUP GUIDES

A GML Markup Guide is a manual which describes the markup conventions and tags for a type (or types) of document. It is essentially a General Document with some specialized elements.

ELEMENT TYPES AND STRUCTURE DEFINITION

This information is presented formally in Figure 21 and Figure 22 on page 149. The definition uses the tokens "word," and "textitem," as well as those on the following list:

bde	Basic Document Element
etd	Element Tag Description
hseg	Heading Segment
ip	Implied Paragraph
markup	Markup Rule
prod	Processing Discussion
sumline	First Line of Tag Summary Entry
tagentry	Tag Summary Entry
ttag	Element Tag
tud	Tag Usage Discussion

The following notation is used:

:	Explicit Tag
=	Attribute
.	Content
,	Sequence
?	Optional
*	Repeatable
/	Alternative
( )	Grouping
[ ]	Restriction

The tag description section was discussed in "Determining Document and Element Types" on page 106 and the colophon in "Element Tag Descriptions" on page 21.

The tag summary is the table in "Appendix A: GML Markup Summary" on page 131. Each entry consists of either a single line (**tagline**) or two lines (a "summary line," then the **tagline**). The lines contain the markup rule from the tag description.

The message list (**ml**) can be found in "Appendix C: GML Messages" on page 141. Each entry consists of a messge number, a message, an explanation, an optional system action, and a user response.

```
1.  :gmgdoc sec=?.frontm, body, appendix?, backm? :e
6.  :body.part*/(hseg1/tagdesc)*
8.  part.h0, bde?*, (hseg1/tagdesc)*
15. bde.list/address/fig/lq/note/p/pcs/xmp/tagsum/ml
29. textitem.figref/fnref/hdref/hp0/hp1/hp2/hp3/ph/q/mv/t/tref
```

Figure 21. Formal Definition of a GML Markup Guide (Part 1 of 2): These elements are variants of General Document elements. The statements are modifications of correspondingly numbered statements in Figure 20 on page 147. The other statements in that figure are also considered part of this definition.

```
Tag Description Section:
1.  :tagdesc.hsegl, colophon?, etd* :e
2.  etd.type, tud, prod*
3.  :type id=? tag=.markup, desc, adesc?*
4.  markup.ttag, attr, attrmore?, cont, terml, term2
5.  tud.usage, txmp
6.  prod.proc, pxmp?
7.  :desc :adesc atag= aname= astatus=?.ip
8.  :usage :proc prof=.ip, bde?* [no fig, fn, xmp]
9.  ttag :attr :attrmore? :cont :terml :term2.word?* [no psc]

Tag Summary:
10. :tagsum.tagentry* :e
11. tagentry.(sumline?, tagline)*

Message List:
12. :ml termhi=? tsize=?.(msgno, msg, xpl, sysact?, uresp)* :e
13. :msg :xpl :sysact :uresp.ip

Text Items
14. :mv.word* :e
15. :t.word :e
16. :tref.(word/textitem)* :e

Generated Element (terminated by end of markup):
17. :colophon.Generated by APF

Text Lines (terminated by end of input line)
18. :msgno.word* [no psc]
19. sumline :tagline.tab, word?*, tab, word?*, tab, word?*, tab, word?*
```

Figure 22. Formal Definition of a GML Markup Guide (Part 2 of 2): Elements Unique to GML Markup Guides.

PERMITTED ATTRIBUTE VALUES

These are the same as for General Documents, plus the following:

aname=	Alphameric character string that is the name of an attribute.
astatus=	Status of an attribute label: 'optional' or 'mandatory'.
atag=	Attribute label: one to eight upper case alphameric characters, with initial alphabetic. (May be entered in lower case.)
prof=	Profile: one to eight upper case alphameric characters, with initial alphabetic. (May be entered in lower case.)
tag=	Element type tag: one to eight upper case alphameric characters, with initial alphabetic. (May be entered in lower case.)

MARKUP CONVENTIONS

These are the same as for General Documents, with the following additional symbols:

&$tab.	Tab character (system symbol).
&adid.	Description of id attribute in tag descriptions.
&tab.	Tab character -- when automatically generating a tag summary.
&tname.	Current element type name in tag description.
&ttag.	Current tag in tag description.
&xmtp.	Example of title page in tag descriptions.

The following are symbols for repetitive phrases used in tag descriptions and elsewhere to maintain consistency.

&nit.	No Immediate Text
&osl.	Text on Same Line
&cpl.	See "Usage" in text
&utl.	Text Item
&cpu.	Paragraph Unit
&gbs.	Generated by APF
&tbs.	Implied P Structure
&ael.	At End of Line
&tbl.	Paragraph or Higher
&tbl2.	Level Element
&eom.	At End of Markup

PROCESSING

The APFs for the tag description elements generate a substantial amount of text. Figure 23 shows the output of the APFs for the markup in Figure 12 on page 107 with attribute values and the content of the type element clearly identified. Content of other elements, and all of the second **adesc** element, was omitted. The optional **astatus** label for the **adesc** element, which was omitted in Figure 12 on page 107, was entered here so that its output could be identified.

$$tag$$ -- $$type content$$

Type	Attributes	Content	Termination
:$$tag$$			

The **$$tag$$** tag identifies

The **$$atag$$** attribute label identifies the $$astatus$$ $$aname$$ attribute. Its value is

Usage: The $$type content$$ can occur

Processing With the "$$prof$$" Profile: The supplied APF

Results:

Figure 23. Output Generated by Element Tag Description APFs

The starter set of GML described in this manual can be used with very complex documents. It provides a variety of powerful functions, ranging from automatic numbering of list items to generating a list of illustrations. You enter descriptive mnemonic tags in your document, such as ":p." for paragraph and ":fig" for figure, and SCRIPT/VS does the rest.

GML is easy to learn and use. But for many documents, you will not need the full set of fifty tags which the starter set provides. This section will get you started with just a few of the tags, with which you can:

1. Produced numbered ("ordered") or bulleted ("unordered") lists automatically.
2. Automatically format headings and a table of contents. And, if you want, you can have GML number your headings using a decimal numbering system. Then, when you add or delete information, the numbering is changed for you.
3. Format text in paragraphs aligned with the current indention level of a list.

GML functions are invoked by using the PROFILE(GDOCPROF) option of the SCRIPT command. If you want automatic heading numbering, you also use the SYSVAR(H NUM) option of the command.

GML TAGS

GML tags identify parts, or "elements," of a document. You always enter a tag at the start of an element; sometimes (we'll tell you when) you also enter a tag to identify when an element ends.

You can get started with just five types of tag:

:hN. Identifies a heading of level N where N is 0 to 6; i.e., :h0., :h1., :h2., :h3., :h4., :h5., or :h6. tag. The heading should be entered with initial capitals, and must fit entirely on the same line as the tag.

If the numbering option is requested, headings of levels 1, 2, 3, and 4 will be numbered 1.0, 1.1, 1.1.1, 1.1.1.1, 1.2, 1.2.1, 1.2.1.1, and so on.

:Tl Identifies a list of type T, where T is the letter "o" (:ol for ordered list), "u" (:ul for unordered list), or "s" (:sl for simple list -- neither bulleted nor numbered). You must end the list with the ":eTl" tag, where T is the same as when you started the list.

Within a list everything is indented. You can start a new list within a list (called a "nested" list) and it will be indented further. When you end the nested list (with the :eTl tag), following text will return to the level of indention of the outer list. You can nest lists for many levels[29] but you always use the same tags regardless of the level of nesting. You can nest different types of list (:ol, :ul, and :sl) within one another.

:li. Identifies an item of a list. In an ordered list (:ol) it will automatically be numbered (or lettered). In an unordered list (:ul) it will be bulleted. In a simple list (:sl) it will be neither numbered nor bulleted. When lists are nested, the style of numbering or bulleting will be different for each level of nesting. (Examples are shown later.) You should not enter the numbers or bullets yourself.

:toc Identifies where a table of contents is to be generated. Headings of level 0, 1, 2, 3, and 4 are included in the table of contents.

:p. Identifies a paragraph. It will be set in block style at the current level of indention.

[29] Within the bounds of a single list the level of nesting cannot be more than 3 for unordered lists, 6 for simple lists, and 6 for ordered lists.

As you can see, all tags begin with a colon (:), which is the GML delimiter character. They all are entered at the beginning of a line. Those tags which end with a period are followed by text on the same line. Each of the other tags is entered on a line by itself.

EXAMPLES

This book was marked up entirely in GML (except for a few control words used in figures), so there are plenty of examples for you to look at. And of course you can look up any tag in the alphabetical listing in "Element Tag Descriptions" on page 21 and find an example there. Figure 24 shows how part of the beginning of this appendix could have been marked up with just the tags we have been discussing. In addition, here are some examples of lists:

```
:p.GML is easy to learn and use.
But for many documents, you will not need the full set of fifty tags
which the starter set provides.
This section will get you started with just a few of the tags, with
which you can:
:ol
:li.Produced numbered ("ordered") or bulleted ("unordered")
lists automatically.
:li.Automatically format headings and a table of contents.
And, if you want, you can have GML number your headings using a
decimal numbering system.
Then, when you add or delete information, the numbering is changed
for you.
:li.Format text in paragraphs aligned with the current indention
level of a list.
:eol
:p.GML functions are invoked by using the PROFILE(GDOCPROF) option
of the SCRIPT command.
If you want automatic heading numbering, you also use the
SYSVAR(H NUM) option of the command.
:h2.GML Tags
:p.GML tags identify parts, or "elements," of a document.
You always enter a tag at the start of an element; sometimes
(we'll tell you when) you also enter a tag to identify when an element ends.
:p.You can get started with just five types of tag:
```

Figure 24. Example of Paragraph, List, and Heading Tags

- Here is an unordered list item at level one.

- And another unordered list item with enough words to show what a lengthy line looks like when formatted.

1. This is item one of a first level ordered list. The list was begun with the following tag (on a line by itself):

   ```
   :ol
   ```

   The list item above was entered on the next input line, as follows:

   ```
   :li.This is item one of a first...
   ```

2. This is item two of a first level ordered list.

   a. This is item one of a second level ordered list. This nested list was begun with the same markup as the outer list:

      ```
      :ol
      ```

      The list item above was entered on the next input line, as follows:

      ```
      :li.This is item one of a second...
      ```

b. This is item two of a second level ordered list. It is long enough to show the "hanging indent" formatting style.

3. This is item three of a first level ordered list.

   This is a paragraph which is part of the list item. It is entered after the text of the :li. tag, and is formatted at the same level of indention. It is marked up the same way inside a list as it would be outside.

   a. This is another item one of a second level ordered list.

      This is a paragraph which is part of a second level list item. It is included to illustrate how the indention is maintained.

      - We can nest an unordered list within an ordered list. Like the other lists, it begins with a list tag on a line by itself:

        ```
        :ul
        ```

        The list item is entered on the following line, just like the others:

        ```
        :li.We can nest an unordered list...
        ```

      - Here is another unordered list item.

        - And of course we can nest still another unordered list within the first one!

          It was entered the same way as the previous ordered list; you always use the same list, list item, and paragraph tags, regardless of the level of nesting.

      1) This is item one of a third level ordered list.

      2) This is item two of a third level ordered list. This added sentence shows how longer items are formatted.

         a) This is item one of a fourth level ordered list. It was entered the same way as ordered list items at higher levels, so we won't show you how again.

            - Here is an unordered list within a fourth level ordered list.

              Of course we can have paragraphs here too.

            - And as many list items as you need.

              - We can even nest another unordered list if the current item requires "sub-bullets."

              - Although we are starting to use so much space for the indention that there is little left for text.

         b) So let's go back to the fourth level.

      3) And the third.

   b. And the second.

4. And back to the first!

This is item one of a level one simple list.

This is item two of a level one simple list.

  This is item one of a level two simple list.

  This is item two of a level two simple list.

    This is item one of a level three simple list.

    This is item two of a level three simple list.

These processing examples are interesting, but the best way to use GML is simply to ignore processing. GML tags are not processing instructions -- they describe the document. Just identify each document element with the tag that describes it, and SCRIPT/VS will take care of the processing.

OTHER FUNCTIONS

With a few more tags, you can have a title page and an appendix section. Just enter the tags shown below. (If you want more information included on the title page -- such as your name -- see Part One of this manual.)

```
:gdoc
:frontm
:titlep
:title.Enter the Title with Initial Capitals Here
:date.Enter the date here (or leave blank and GML will provide it)
:etitlep
:toc
:body
(Enter your tags and text here.)
:appendix
(Enter more tags and text if you like -- each :h1. starts
another appendix.)
:egdoc
```

Here's what these tags do:

:gdoc	Identifies your document as a "General Document," rather than some other type which your installation may also support.[30] You must end a General Document with the ":egdoc" tag.
:frontm	Identifies the front matter, which appears ahead of the main text. Front matter pages are numbered with Roman numerals.
:titlep	Identifies the title page. You must end the title page with the ":etitlep" tag.
:title.	Identifies the document title. Enter it entirely on the same line, with initial capitals.
:date.	Identifies the document date. You can omit the date (but not the tag!) and the system will supply the date.
:body	Identifies the main portion of the document (and implicitly ends the front matter). Page numbering starts again at page one, but in Arabic numbers.
:appendix	Identifies the appendix section (and implicitly ends the body). In the appendix section, each :h1. tag starts a new appendix on a new page. The words "Appendix X:" are prefixed to the heading, where X is the current appendix letter. The appendix section is optional. Omit the tag if the document has no appendixes.

The full starter set also offers tags for cross-references to headings, figures, and footnotes, and tags for emphasizing phrases. It also has tags for definition lists, such as the one that precedes this paragraph. You'll find those other tags -- thirty in all -- just as easy to learn as the twenty you've already learned about.

[30] With GML you can define the exact tags and processing you need for each type of document produced at your installation. The starter set is just an example of support for a particular type (although a very general one).

Glossary terms are defined as they are used in this book. Element type names and tags, attribute names and labels, and SCRIPT command options are, for the most part, not included in the glossary. Their definitions can readily be found in the alphabetic listing of tag descriptions in Part One of this manual, and in the reference summaries in Appendixes A and B. If you cannot find the term you are looking for, refer to the table of contents, the index, or to the IBM Data Processing Glossary, GC20-1699.

ampersand: The "&" character.

When an ampersand begins a character string, SCRIPT/VS assumes the character string is a symbol name. If the symbol name is defined, SCRIPT/VS replaces the symbol with its value (unless symbol substition is off).

APF: Application processing function.

application processing function (APF): In GML processing, the processing that is performed when a document element or attribute is recognized. In SCRIPT/VS, an APF is implemented as a sequence of control words, possibly intermixed with text and symbols, in one of three forms: macro definition, value of a symbol, or imbedded file.

attribute: A characteristic of a document (or document element) other than its type or content. For example, the security level of a document or the depth of a figure.

attribute label: In GML markup, a name of an attribute that is entered in the source document when specifying the attribute's value.

back matter: In a book, those sections (such as glossary and index) that are placed after the main chapters or sections.

balancing: In multicolumn formatting, the process of making column depths on a page approximately equal.

basic document element: In a General Document, one of a group of elements which occur frequently. They are: note, paragraph, paragraph continuation, definition list, ordered list, simple list, unordered list, address, example, figure, and long quotation.

batch environment: The environment in which non-interactive programs are executed.

binding edge: The edge of a page to be bound, stapled, or drilled. Defined with the BIND option of the SCRIPT command.

body: (1) Of a printed page, that portion between the top and bottom margins that contains the text. (2) Of a book, that portion between the front matter and the back matter.

boldface: A heavy-faced type. Also, printing in this type.

bug: An error in a program or in document markup.

caps: Capital letters. (See also initial caps.)

caption: Text accompanying and describing an illustration.

chapter: In a General Document, a name given to a first-level heading segment that occurs within the body of of document. (See also heading segment.)

character: A symbol used in printing. For example, a letter of the alphabet, a numeral, a punctuation mark, or any other symbol that represents information.

Cicero: In the Didot point system, a unit of 0.1776 inches (4.512 millimeters) used in measuring typographical material.

CMS: An interactive processor that operates within VM/370.

command: A request from a terminal or specified in a batch processing job for the performance of an operation or the execution of a particular program. For example, a request given at a terminal for SCRIPT/VS to format a document, or for an editor to edit a line of text.

composition: The act or result of formatting a document.

concatenation: The forming of an output line that contains as many words as the column width allows, by placing the first words from an input line after the last words from the preceding input line. When words from an input line would reach beyond the right margin and hyphenation cannot be performed, they are placed at the beginning of the next output line, and so on.

content: A primary characteristic of a document (or element) that defines its component parts. In markup, the content may be immediate text, or it may consist of elements that are tagged explicitly.

control word: An instruction within a document that tells SCRIPT/VS how to process the document. (See also macro.)

control word line: An input line that contains at least one control word.

current line: The line in a source document at which a computer program (such as an editor or a formatter) is positioned for processing.

debug: To detect, trace, and eliminate errors in computer programs and SCRIPT/VS documents.

default value: A value assumed by a computer program when a control word, command, or control statement with no parameters is processed. In GML processing, the value assumed for an attribute when none is specified.

dictionary: A collection of "word stems" that is used with the spelling verification and automatic hyphenation functions.

Didot point system: A standard printer's measurement system on which type sizes are based. A Didot point is 0.0148 inches (0.376 millimeters). There are 12 Didot points to a Cicero. (See also Cicero and point.)

document: (1) A publication or other written material. (2) A machine-readable collection of lines of text or images, usually called a source document. (See also output document and source document.)

document administrator: One who is responsible for defining markup conventions and procedures for an installation. This also involves creating document type descriptions, and specifying the processing to be performed on the documents.

document type: See type.

document type description: An unambiguous record of the characteristics of a type of document. It includes a definition of the content structure (formal or informal), permitted attribute values, allowable symbols, and other information that will assist in preparing GML Markup Guides and developing APFs. (See also formal definition.)

duplex: A mode of formatting appropriate for printing on both sides of a sheet.

edit: To create or modify the contents of a document or file. For example, to insert, delete, change, rearrange, or copy lines.

editor: A computer program that processes commands to enter lines into a document or to modify it.

element: A component part of a document. It is actually a document which is part of the content of a larger document. Being a document, it has a type and a content, and may have attributes as well. (See also content and structure.)

element type: See type.

em: A unit of measure for a particular font that is equal to the point size of that font. In a font that is not proportionally-spaced, an em is equivalent to a character.

float: (1) (noun) A keep (group of input lines kept together) whose location in the output document and printed page may vary from its location in the source document. (2) (verb) Of a keep, to be formatted in a location different from its location in the source file.

flush: Having no indention.

font: An assortment of type, all of one size and style.

footing: Words located at the bottom of the text area. (See also running footing.)

footnote: A note of reference, explanation, or comment, placed below the text of a column or page, but within the body of the page (above the running footing).

foreground: The environment in which interactive programs are executed. Interactive processors reside in the foreground.

formal definition: A diagram which uses a combination of tags, attribute labels, and special characters to define precisely the structure of a document. For each type of element in the document, it shows where the element can occur, what attributes it has, how it is to be marked up, and the structure of its content. (See also document type description.)

format: (1) (noun) The shape, size, and general makeup of a printed document. (2) (verb) To prepare a document for printing in a specified format.

formatter: A computer program that prepares a source document to be printed.

front matter: In a book, those sections (such as preface, abstract, table of contents, list of illustrations) that are placed before the main chapters or sections.

General Document: A type of document whose description can apply to a variety of documents, from memoranda to technical manuals. It may be used as a "catch-all" category for documents which do not conform to any other type description.

Generalized Markup Language (GML): A language for describing the characteristics of a document without respect to particular processing.

generic element: An element whose type suggests only the structural role the element plays in the document, rather than the semantic role as well. Compare the General Document "ordered list," for example, with a "removal instructions" element that might be part of a "service manual" document type. Both suggest a numbered list, but the second tells what it is a list of.

GML: Generalized Markup Language

GML delimiter: A special character that denotes the start of GML markup. In the starter set, it is initially a colon (:).

GML interpretation: Interpretation of GML markup consists of recognizing the start or end of an element (or an attribute label), associating it with an APF, and executing the APF. In SCRIPT/VS, interpretation is performed jointly by SCRIPT/VS itself and by APFs.

GML Markup Guide: A manual which describes the markup conventions and tags for a type (or types) of document. As a document type, it is essentially a General Document with some specialized elements.

GML type: See type

graphic PSC element: A PSC element that produces permanent formatting effects not conveniently possible with available APFs.

hanging indention: The indention of all lines of a block of text, following the first line (which is not indented the same number of spaces).

heading: Words located at the beginning of a chapter or section or at the top of a page.

heading segment: An element which begins with a heading, followed by basic document elements and lower-level heading segments.

hexadecimal: Pertaining to a number system based on 16, using the sixteen digits 0, 1, . . . 9, A, B, C, D, E, and F. For example, hexadecimal 1B equals decimal 27.

highlighting: Emphasis associated with a document element. In formatting, highlighting is usually expressed by changing font, underscoring, and/or capitalizing the highlighted element.

implied paragraph structure: An element which begins with an "implied" paragraph; that is, one for which you do not enter a paragraph tag. The existence of the paragraph is understood from the existence of the implied paragraph structure.

indent: To set typographical material to the right of the left margin.

indention: The action of indenting. The condition of being indented. The blank space produced by indenting.

initial caps: Capital letters occuring as the first letter of each word in a phrase. To set a phrase in initial caps is to capitalize the first letter of each word in the phrase.

input device: A machine used to enter information into a computer system (for example, a terminal used to create a document).

input line: A line, as entered into a source file, to be processed by a text processor.

interactive: Pertaining to an application in which entries call forth a response from a system or program, as in an inquiry system. An interactive system might also be conversational, implying a continuous dialog between the user and the system. Interactive systems are usually communicated with via terminals, and respond immediately to commands. (See also foreground.)

interactive environment: The environment in which an interactive processor operates.

italic: A typestyle with characters that slant upwards to the right.

JCL: Job control language.

job control language (JCL): A language of control statements used to identify a computer job or describe its requirements to the operating system.[31]

job control statement: A statement that provides an operating system with information about the job being run.

justify: To insert extra blank space between the words in an output line to cause the last word in the line to reach the right margin. As a result, the right-hand edge of each output line is aligned with preceding and following output lines.

[31] American National Dictionary for Information Processing

keep: (noun) In a source document, a collection of lines of text to be printed in the same column. When the vertical space remaining in the current column is insufficient for the block of text, the text is printed in the next column. (In the case of single-column format, the next column is on the next page.)

layout: The arrangement of matter to be printed. (See also format.)

lower case: Pertaining to small letters as distinguished from capitals; for example, "a, b, g" rather than "A, B, G."

machine-readable: Ability of data to be acquired or interpreted by a machine, from a storage device, from a data medium, or from another source.

macro: An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language. In SCRIPT/VS, a macro definition is a sequence of one or more input lines that can contain control words, symbols, text, and GML markup.

margin: (1) The space above, below, and on either side of the body of a page. (2) The left or right limit of a column. (See also **Figure reference 'Apage' unresolved**.)

mark up: (verb) (1) To determine the markup for a document. (2) To insert markup into a source document.

markup: (noun) Information added to a document that enables a person or system to process it. Markup may describe the document's characteristics, or it may specify the actual processing to be performed. In SCRIPT/VS, markup consists of GML tags, attribute labels and values, and control words.

markup editor: One who marks up individual documents according to an installation's conventions and procedures established by the document administrator.

normal form: The normal form of a document is that which is reflected in the published copies, and expressed in the document type description.

option: Information entered with the SCRIPT command to control the execution of SCRIPT/VS.

output device: A machine used to print, display, or store the result of processing.

output document: A machine-readable collection of lines of text or images that have been formatted, or otherwise processed, by a document processor. The output document can be printed or it can be filed for future processing.

output line: A line of text produced by a text processor.

paginate: To number pages.

paragraph unit: An element that has the same structure as a paragraph. In a General Document, the paragraph units are: paragraph, note, and paragraph continuation.

parameter: Items of data entered on the same line as a control word which govern the control word's behavior.

part: In a General Document, a part is a zero-level heading segment. (See also heading segment.)

patch PSC element: A PSC element that is used temporarily to modify the normal output; for example, to prevent a widow.

pica: A unit of about 1/6 inch used in measuring typographical material. Similiar to a Cicero in the Didot point system.

point: (1) A unit of about 1/72 of an inch used in measuring typographical material. There are twelve points to the pica. (2) In the Didot point system, a unit of 0.0148 inches. There are twelve Didot points to the cicero.

process-specific controls element (PSC element): An element which contains control words and text which are meaningful for one or more processes that will be performed on the document. PSC elements are the recommended convention for entering control words that would otherwise disturb the generality of a document.

profile: In SCRIPT/VS processing, a file that is imbedded before the primary file is processed. It can be used to control the formatting of a class of source documents. When processing GML markup, the profile usually contains the mapping from GML to APFs, and the symbol settings that define the formatting style.

profile PSC element: A PSC element that supplements the profile document; for example, by defining additional symbols.

proportional spacing: The spacing of characters in a printed line so that each character is alloted a space proportional to the character's width.

PSC element: Process-specific controls element.

ragged right: The unjustified right edge of text lines. (See also justify.)

reference element: In a General Document, an element whose content is a reference to another element that is generated by an APF. There are three: figure reference, footnote reference, and heading reference.

required blank: A character that prints as a blank, but does not act as a word separator.

right-hand page: The page on the right when a book is opened; usually odd-numbered.

rule: (noun) A straight horizontal or vertical line used, for example, to separate or border the parts of a figure or box.

running footing: A footing that is repeated above the bottom margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the page's body (text area).

running heading: A heading that is repeated below the top margin area on consecutive pages (or consecutive odd- or even-numbered pages) in the page's body (text area).

source document: A machine-readable collection of lines of text or images that is used for input to a computer program.

space: A blank area separating words or lines.

space unit: A unit of measure of horizontal or vertical space. In GML markup, the em is used when a measure that is relative to the current font size is required. When an absolute measure is required, as in specifying the depth of a figure, recommended space units are inches (nnI), millimeters (nnW), picas/points (nnPnn), or Ciceros/Didot points (nnCnn), where "nn" is the number of units. (See also em, pica, point, Cicero, and Didot point system.)

starter set: An example of GML support that is provided with the Document Composition Facility. It consists of a document type description for "General Documents," a profile, and a library of APFs.

structure: A characteristic of a document (or element) that expresses the type and relationship of the elements of the content. (See also content and element.)

symbol: A name in a source document that can be replaced with something else. In SCRIPT/VS, a symbol is replaced with a character string. SCRIPT/VS may interpret the character string as a numeric value, a character string, a control word, or another symbol.

symbol substitution: During formatting, the replacement of a symbol with a character string which SCRIPT/VS may interpret as a value (numeric, character string, or control word) or as another symbol.

SYSVAR: An option of the SCRIPT command which permits the user to specify values for symbols. In the starter set, SYSVAR symbol values determine whether certain processing variations will occur, such as heading numbering, duplex formatting, and two-column printing.

tab: (1) (noun) A preset point in the typing line of a typewriter-like terminal. A preset point in an output line. (2) (verb) To advance to a tab for printing or typing. (3) a tab character, hexadecimal code X'05'.

tag: In GML markup, a name for a type of document (or document element) which is entered in the source document to identify it. For example, ":p." might be the tag used to identify each paragraph.

terminal: A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

text item: Explicitly marked ("tagged") elements that occur within text, such as within a paragraph unit. In a General Document, for example, quotes and phrases are text items.

text line: An input line that contains only text.

text programmer: One who implements APFs that provide the processing specified by the document administrator. In SCRIPT/VS, this involves writing SCRIPT/VS macros and organizing macro libraries and profile files so that the appropriate composition will be done for each tag.

TSO: An interactive processor within OS/VS2.

type: A primary characteristic of a document (or element) that expresses its role or purpose. It is literally an answer to the question: "What type of document is this?." "Type" is sometimes referred to as "document type," "element type," or "GML type."

typeface: All type of a single style. There might be several fonts (different sizes) with the same typeface or style.

typeset: (1) (verb) To arrange the type on a page for printing. (2) (adjective) Pertaining to material that has been set in type.

underscore: (1) (noun) A line printed under a character. (2) (verb) To place a line under a character. To underline.

unique identifier (ID): In a General Document, an attribute whose value serves as a name which can be used to refer to the element. (See also reference element.)

upper case: Pertaining to capital letters, as distinguished from small letters; for example, "A, B, G" rather than "a, b, g."

widow: A single output line that is printed in a different column from the text with which it is associated so as to create a typographically unacceptable effect. For example, A line of a paragraph that is printed separately from the rest of the paragraph, or a heading that is separated from the section it heads.

word spacing: The space between words in a line. Also called wordspace or interword space.

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Document Composition Facility:
Generalized Markup Language (GML)
User's Guide
SH20-9160-0

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ________________

Previous TNL ________________

Previous TNL ________________

Fold on two lines, tape, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

SH20-9160-0

Reader's Comment Form

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Tape

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Document Composition Facility: Generalized Markup Language (GML) User's Guide Printed in U.S.A. SH20-9160-0

Document Composition Facility:
Generalized Markup Language (GML)
User's Guide
SH20-9160-0

Reader's Comment Form

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

List TNLs here:

If you have applied any technical newsletters (TNLs) to this book, please list them here:

Last TNL ____________________

Previous TNL ________________

Previous TNL ________________

Fold on two lines, tape, and mail. No postage necessary if mailed in the U.S.A. (Elsewhere, any IBM representative will be happy to forward your comments.) Thank you for your cooperation.

SH20-9160-0

Reader's Comment Form

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and Tape

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601

Document Composition Facility: Generalized Markup Language (GML) User's Guide Printed in U.S.A. SH20-9160-0

SH20-9160-0

Document Composition Facility: Generalized Markup Language (GML) User's Guide Printed in U.S.A. SH20-9160-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, N.Y. 10604

IBM World Trade Americas/Far East Corporation
Town of Mount Pleasant, Route 9, North Tarrytown, N.Y., U.S.A. 10591

IBM World Trade Europe/Middle East/Africa Corporation
360 Hamilton Avenue, White Plains, N.Y., U.S.A. 10601