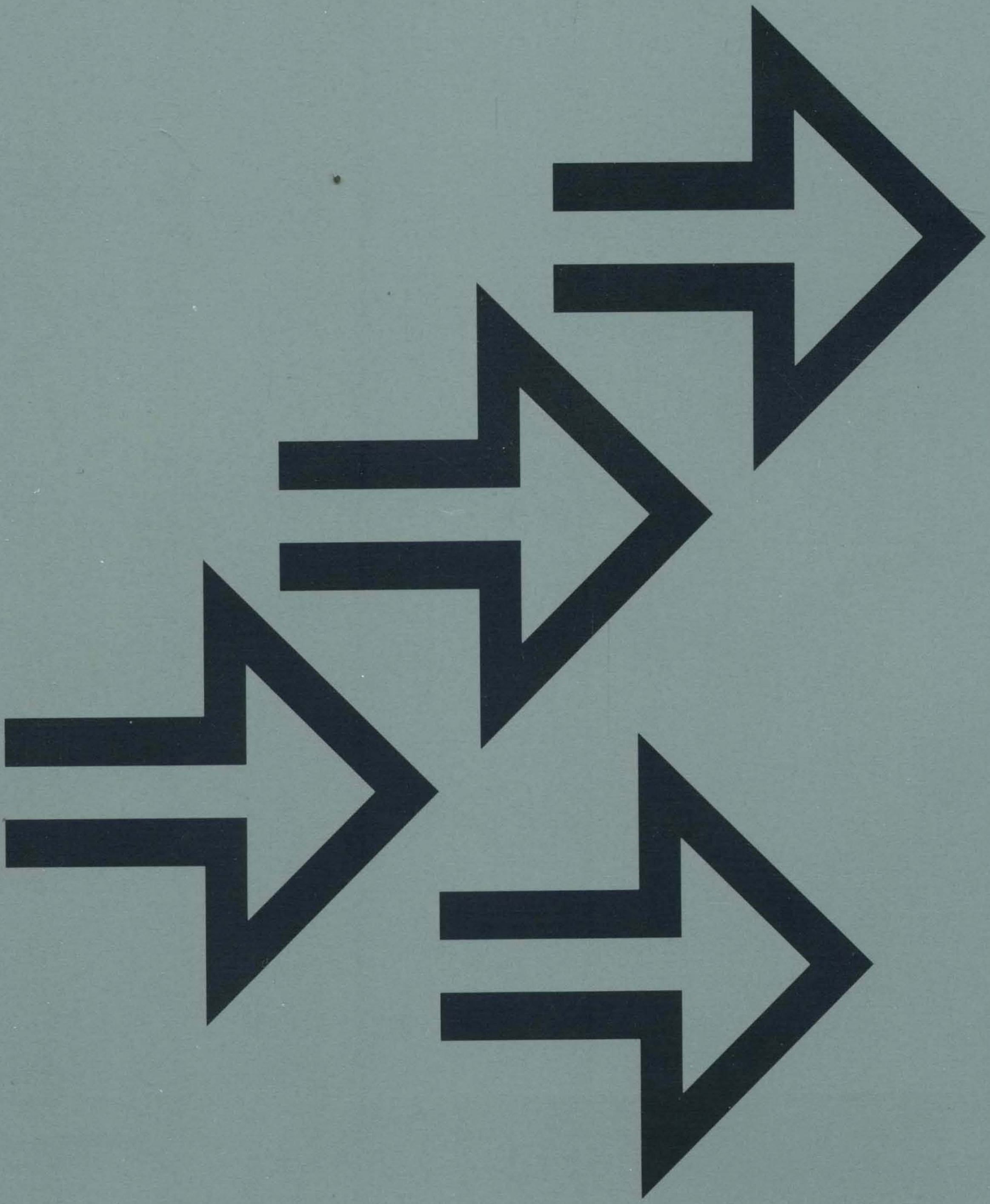
IBM

Document Composition Facility Generalized Markup Language Implementation Guide

Program Product

SH35-0050-2



Program Number 5748-XX9

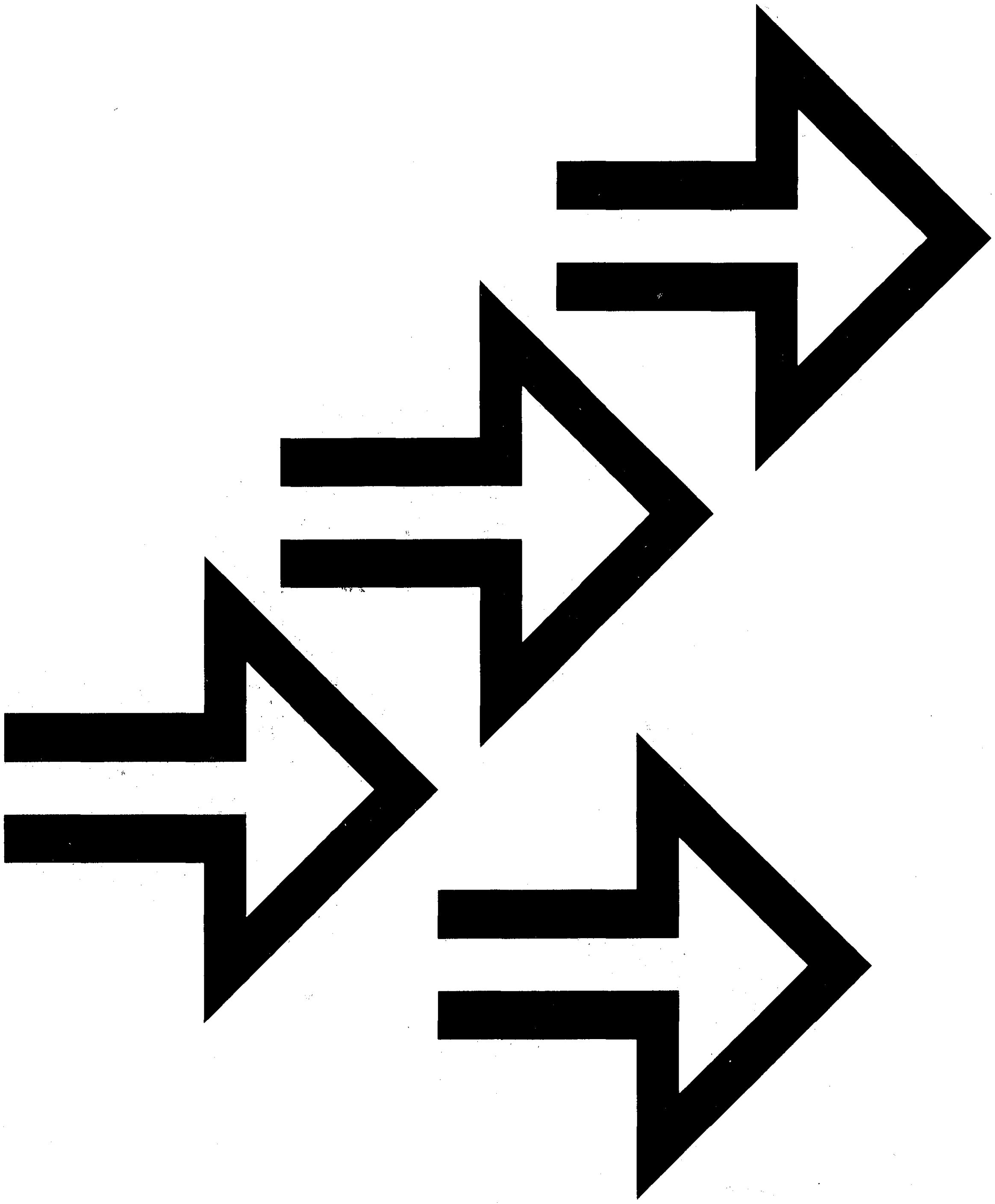
Release 3

IBM

Document Composition Facility Generalized Markup Language Implementation Guide

Program Product

SH35-0050-2



Program Number 5748-XX9

Release 3

This publication was produced using the IBM Document Composition Facility (program number 5748-XX9).

Third Edition (March 1985)

This is a major revision of SH35-0050-1, which is now obsolete. Significant changes are summarized under "Summary of Amendments" following the "List of Illustrations".

This edition applies to Release 3 of the Document Composition Facility program product, Program Number 5748-XX9, and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

Preface

The Generalized Markup Language (GML) is a language for document description. It can be used to describe the structure and text elements (parts) of a document without regard to the processing that may be required to format them. The GML starter set is an implementation of GML written in the SCRIPT/VS formatting language. It is distributed with the Document Composition Facility program product (Program Number 5748-XX9) and is provided as an example of a GML application for a *general document*.

The GML starter set consists of the following components:

- GML tags which describe a *general document*
- A macro library[1]
- A profile (DSMPROF3).[2]

Purpose of the Book

The purpose of this book is to assist text programmers who are responsible for maintaining, altering, or extending the GML starter set[3]. This book provides specific, detailed information about how the starter set tags work. It should also be of substantial use to those who need to write their own GML tags and Application Processing Functions (APFs) independent of the starter set. (It is not intended for end users of either the GML starter set or SCRIPT/VS control words.)

Modifying the APFs requires an understanding of the way the starter set works, how and why it was written the way it is, and the symbols and conventions used in writing the APFs.

Developing a GML application involves several basic steps. These include:

- Defining the type of document to which the GML applies
- Identifying the structure and text elements of the document
- Defining tag names for the text elements and structure of the document
- Defining any attributes required to fully describe the elements
- Identifying the formatting that should be performed for each document element

---

[1] The name of the library is different for each operating environment. In CMS the library is named DSMGML3 MACLIB. In TSO the library is a partitioned data-set named SCRIPT.R30.MACLIB. In ATMS the macros reside in the GML SYSOP's permanent storage and in DLF the macros reside in public library 1314151.

[2] In CMS the profile is in a file named DSMPROF3 SCRIPT. In the other operating environments the profile is also named DSMPROF3 and is part of the macro library.

[3] The GML starter set is a fully supported part of the Document Composition Facility program product provided that neither the profile nor the macro library have been modified in any way. What this means is that you should be careful to not alter the base version of these files but rather should make your own copies or user libraries.

- Writing APFs and macros that will provide the desired formatting
- Writing a profile that establishes the formatting environment, associates the tags with the appropriate APFs, and enables the GML services in SCRIPT/VS.

This book describes how we did the last two steps in the process described above for the starter set. The other steps are discussed in the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* and the *Document Composition Facility: SCRIPT/VS Language Reference.*

Prerequisites

This book assumes that the reader is familiar with the GML starter set distributed with the Document Composition Facility (DCF) and the use of SCRIPT/VS control words. Knowledge of the internal functions of the starter set or SCRIPT/VS is not required.

Particular areas of SCRIPT/VS that the reader should be familiar with are:

- Conditional processing
- Using symbols
- Writing SCRIPT/VS macros
- GML support in SCRIPT/VS.

Each of these topics is the subject of a chapter in the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide.* If you are not familiar with these subjects, you might find it useful to review them before proceeding.

You should also consider reviewing the following topics:

- The control word modifier (')
- Word continuation.

How to Use this Book

Most APFs are discussed on a line-by-line basis. "Appendix C. Starter Set Macro Library Listing" on page 181 contains a complete listing of the GML macro library and the DSMPROF3 profile for reference as you read.

The starter set uses many SCRIPT/VS control words. Because we do not explain all of them in detail in this book, it's a good idea to have a copy of the *Document Composition Facility: SCRIPT/VS Language Reference* available as a reference.

At the end of most chapters in this book, there are some suggested modifications that can be made to the APFs discussed in the chapter. We recommend that you try some of these changes when you finish the chapter. This will help to clarify the way the APFs work together and will give you experience in working with the starter set.

Once the general concepts and approach are clear, this book can probably be best used as a reference manual. When you want to modify a particular aspect of the starter set, just review the chapter that pertains to it and check the index for the symbols you are interested in to see where else they are used. You will want to be sure that if you change a symbol, that the change is appropriate for all the uses of the symbol throughout the library.

Organization of this Book

The first two chapters of the book provide an introduction to GML, GML processing in SCRIPT/VS, special techniques used in the starter set and general information about how the starter set was written.

The remainder of the book is divided into eleven major chapters, each one focusing on a specific function provided by the starter set. The functional areas covered are:

- Starter Set Initialization
- Title page
- Document sections
- Headings
- Paragraphs
- Lists
- Examples and figures
- Quotes, notes, footnotes, and highlights
- Indexing
- Cross referencing
- Miscellaneous.

Behind each GML tag in the starter set is one or more macros and Application Processing Functions (APFs). Each macro used is described in line-by-line detail in this book. Each chapter:

- Explains why each macro (APF) is constructed the way it is
- Explains how the macros relate to each other in the macro library
- Contains suggestions and examples of ways to modify the starter set and guides you through illustrations for:
  - Tailoring the starter set to an installation's publications standards,
  - Enhancing specific APFs to provide specialized formatting functions, and
  - Designing new GML applications.

"Appendix A. Modifying the Macros" on page 171 reviews the steps necessary to update the starter set macros in each of the environments that the Document Composition Facility works: CMS, TSO, ATMS-III, and DLF.

"Appendix B. Migration from Release 2 to Release 3" on page 177 discusses some of the steps that were necessary to update the starter set to support page printers. It is designed as a guide to users who are converting their own GML application to use Release 3 of DCF and to format for page printers.

"Appendix C. Starter Set Macro Library Listing" on page 181 contains a listing of all the macros in the GML starter set macro library.

The index at the back provides references for each global symbol used in the starter set, as well as subject, macro, and tag entries. For all the APF entries, the first one listed is the page on which the APF is described in detail. The other page numbers are secondary references.

Related Publications

These are the publications you will want to have available to you that describe the SCRIPT/VS control words:

- *Document Composition Facility: SCRIPT/VS Text Programmer's Guide*, SH35-0069
- *Document Composition Facility: SCRIPT/VS Language Reference*, SH35-0070

If you are using the Document Library Facility (DLF), you will also need to have the following book available:

- *Document Library Facility Guide*, SH20-9165.

The end-user portion (the tags) of the starter set is described in these publications:

- *Document Composition Facility: GML Starter Set User's Guide*, SH20-9186
- *Document Composition Facility: GML Starter Set Reference Manual*, SH20-9187.

Table of Contents

List of Illustrations

Summary of Amendments

Third Edition

This revision includes minor technical and editorial changes, and changes made to the GML starter set to support the IBM 3820 Page Printer.

Second Edition

This revision includes minor technical and editorial changes, and changes made to the GML starter set to support the 3800 Printing Subsystem Model 3.

A list of the technical changes follows in order by chapter.

Preface

In the section entitled "Related Publications" on page vi a reference to the *Document Library Facility Guide* has been added.

Starter Set Initialization

In the section entitled "The DSMPROF3 Profile" on page 20 the mapping of APFs for the index tags is more fully explained.

In the section entitled "DSM#STYL" on page 32 the processing for offset layout has been changed.

Headings

A new section called "Formatting Special Characters in Headings" on page 85 has been added to explain how to get special characters and font changes into headings.

Paragraphs

In the section called "DSMPARA5" on page 89 the processing done by the DSMPARA5 macro has been expanded.

Quotes, Notes, Footnotes and Highlights

A new section called "Printing Footnotes at the End of a Chapter" on page 134 has been added to explain a technique to place footnotes at the end of the chapter.

Introduction

Basic Concepts

Before looking at the details of the starter set, let's review some basic concepts of Generalized Markup Language (GML) and then look at how it all works together when SCRIPT/VS processes GML markup.

Document Types

There are many different kinds of documents, such as:

- Memos
- Letters
- Insurance policies
- Novels
- Instruction manuals
- Recipe books
- *General documents.*

Each kind of document has its own types of text elements and its own document structure. Document structure refers to how the document is put together into sections such as *front matter*, *body*, *back matter* and so on. Some documents may not have a front matter section or back matter section. Some may have only body matter.

It is the job of a text administrator to identify the document type and the text elements that go with it. Once these have been defined, tags must be assigned to identify each element. The last step is to write Application Processing Functions (APFs) to interpret the tags and perform the formatting desired for the text element.

General Document: The GML starter set includes a set of tags that describe a *general document*. These tags are not designed to meet all text processing needs or to describe all possible text elements. Additional tags will be necessary to produce different kinds of documents.

Because most documents contain basic text elements such as paragraphs, lists, and figures, the starter set application can be used as a base upon which you can build your own GML application.

Tags

A tag is a name for a particular text element or structure found in a document. Each different type of document element should have a unique tag to identify it. For example, a paragraph is a text element and should have a tag to identify it. An item in a list may also be a paragraph, but it is primarily a *list item.* There may even be several paragraphs within a list item. The tag identifying list items should be different from the tag identifying paragraphs.

The formatting to be performed for a specific text element is independent of the tag name. A paragraph is a paragraph regardless of what part of the document it is in or whether it is indented, hanging, or capitalized. The tag (:P in the starter set) simply identifies the text that follows as a paragraph.

Attributes

Some text elements have changeable characteristics. In this case, rather than define two tags, attributes can be used to further describe the text elements. Tags identify the text element and attributes identify particular qualities or characteristics of the element. For example, a figure that is set off from the text by a horizontal rule is essentially the same element as a figure with a box around it. Therefore, there is only one tag for figures, :FIG, and the attribute, FRAME, identifies the type of frame. It is important to understand that the attribute is *not* describing the formatting of the frame, it is identifying the *type* of frame to be used for the figure.

Profile

A profile is a file that contains general information about what processing is to be performed on the source (text) document. The profile, which is specified on the SCRIPT command, is processed before the source file. Its primary purpose is to establish the formatting and processing environments. The profile (along with macros it calls) defines the page layout for the document and the style of formatting for such things as headings and lists. It also establishes the GML rules for scanning for tags and defines the tags and APFs for the application.

DSMPROF3 is the profile for the Document Composition Facility Release 3 starter set. We'll look at the profile in more detail later.

Epifile

An epifile is processed after the source document has been processed. It is physically part of the same file as the profile. The profile portion ends with a .EF [End of File] control word. What follows the .EF [End of File] is, by definition, the epifile. The epifile can be used to provide any processing necessary after the formatting of the document, or perhaps more significantly, between the first and second formatting passes.

In the case of the starter set, the epifile produces the cross-reference listing of IDs, the imbed trace, and the SYSVAR 'W' file of IDs if these have not already been produced by a :EGDOC tag.

APFs

Application Processing Functions (APFs) provide the formatting and processing instructions that are to be performed for a specific tag. The APF behind a tag can be a control word, a symbol or a macro. In the starter set, the APFs are macros which reside

- In the DSMGML3 macro library in the CMS environment,
- In the permanent storage of the GML SYSOP operator in ATMS-III,
- In a partitioned data-set in TSO, and
- In a public library in DLF.

In the profile, a tag is associated with an APF using the .AA [Associate APF] control word. When a tag is encountered, SCRIPT/VS invokes the APF associated with it. If no APF has been

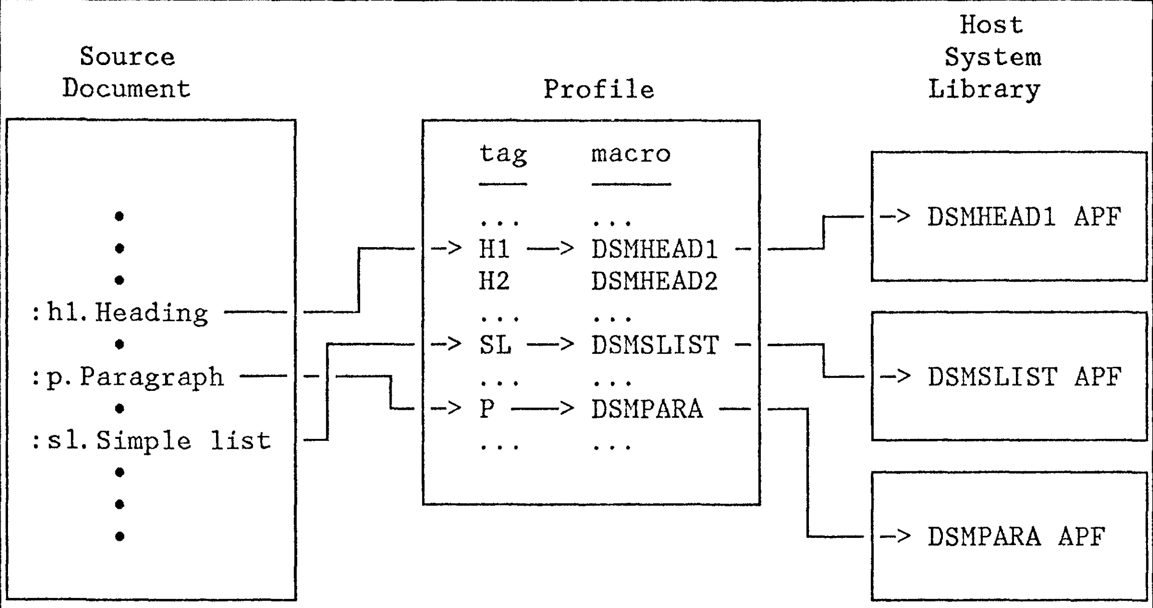


Figure 1. Processing Documents with GML: The profile provides the mapping between tags, which identify elements of text in the source document, and APFs, which provide formatting functions.

associated with the tag, SCRIPT/VS searches for a macro or control word with the same name[4] as the tag, and if found, uses it. If neither a macro nor control word is found, an error message is issued.

APFs can contain SCRIPT/VS control words, symbols text, and macro calls. The text that belongs to the tag is available to the APF for processing by using the .GS [GML Services] SCAN control word. The macros to process the attributes can be invoked using the .GS [GML Services] EXATT control word.

Control Words

Control words are specific instructions to SCRIPT/VS. They do not identify text elements, but rather instruct SCRIPT/VS how the page is to be set up and how the text that follows is to be formatted.

SCRIPT/VS control words are always two characters long (such as "SP" or "BX"). They are always preceded by a period and need to start in column one of the input line.[5]

How Does GML Work?

Within SCRIPT/VS there are numerous mechanisms that make it possible to create and process GML. The more you understand about how SCRIPT/VS processes GML tags, symbols and macros, the better able you will be to understand and modify the starter set and write your own GML tags.

[4] Macro names are not case sensitive. In other words, DSMFIG is the same as dsmfig.

[5] It is possible to create "logical" input lines by using the control word separator. See *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for more details on control words.

Profile Processing

SCRIPT/VS allows a profile to be specified on the SCRIPT command with the PROFILE option.

```
SCRIPT GMLSSDOC (PROF(DSMPROF3)
```

By specifying a file that will be processed before our source document, we can get everything set up the way we want it. This includes defining tags and telling SCRIPT/VS how to find the processing instructions for the tags.

The most important actions that the profile performs are:

- Defines the delimiters for tags (:) and end tags (:e)

  ```
  .dc gml : : e
  ```

- Defines the tags and their associated APFs, for example

  ```
  .aa h1 DSMHEAD1
  ```

  defines a tag named "h1" and associates it with an APF named DSMHEAD1. Attribute rules, which are discussed below, may also be given on the .AA [Associate APF] control word.

- Turns on scanning for tags.

  ```
  .gs tag on
  ```

- Defines the attribute rules for each tag.

  ```
  .gs rules (att novat stop nomsg) (noatt)
  ```

  where

att	specifies that labelled attributes are allowed
novat	specifies that no value attributes are allowed
stop	specifies that when an invalid attribute is found during the scan, the scan is stopped at that point
nomsg	prevents a message from being issued when an invalid attribute is found
noatt	specifies that no attributes are allowed on the end tag.

  The list within the first set of parentheses on the .GS [GML Services] RULES control word sets the rules for the start tags and the list within the second set of parentheses sets the rules for the end tags. There are rarely attributes on an end-tag.

  The definitions of the tags on the .AA [Associate APF] control word may override the general attribute rules specified with the .GS [GML Services] control word. If a particular attribute rule has been defined in the profile using the .GS [GML Services] RULES control word and a contradictory rule is also defined with an .AA [Associate APF] control word, the .AA [Associate APF] control word specification is used. For example, in the starter set the :OL tag is defined as

  ```
  .aa ol dsmlistm (vat) dsmelist
  ```

  Since the general rules allow attributes (ATT), the :OL tag definition does not need to. However, the general rules also specified that value attributes are not allowed (NOVAT) so the :OL tag has to do something to allow value attributes (VAT) in order to be able to process the COMPACT and BREAK attributes on this tag.

The general rules for end-tags specify that there are no attributes on end-tags. There are no end-tags in the starter set which have attributes so this rule is never over-ridden on the tag definitions. See the portion of the DSMPROF3 profile which specifies the .AA [Associate APF] control word definitions for the tags.

We'll look at what else DSMPROF3 does in "Starter Set Initialization" on page 19.

Scanning for Tags and Attributes

Let's take a look at how tags and attributes are processed by SCRIPT/VS.

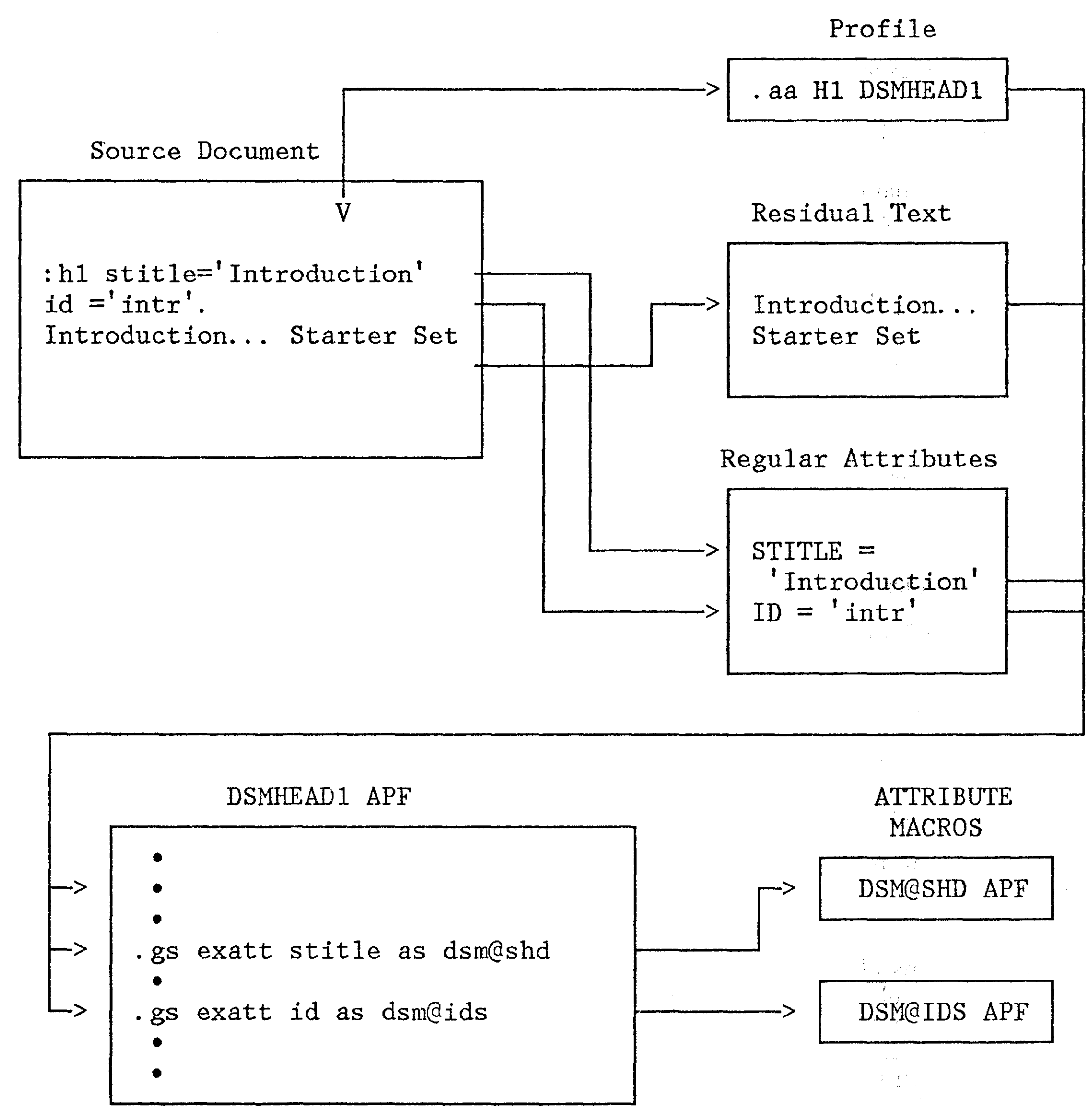


Figure 2. SCRIPT/VS GML Processing.: This figure shows the automatic processing that occurs when SCRIPT/VS encounters a :H1 tag.

As SCRIPT/VS reads each line of the input file, it looks for GML tags. It knows how to find them because the profile defined the tag delimiters with

```
    .dc gml : : e
```

SCRIPT/VS can tell from the delimiter whether a tag is a start tag or an end tag. This is important because they are usually processed differently. Notice that using ":e" as the end-tag delimiter implies that there can be no tag names that start with "e." If you did start a tag name with an "e," SCRIPT/VS would assume it was an end-tag and would always associate the tag with the APF specified as the end-tag APF.

SCRIPT/VS also knows where the tag name ends and the attributes or text begin because the tag name is followed by either a blank or a period.

```
   :h1 stitle='Introduction'
   id='intr'.
   Introduction to the GML Starter Set
   :ol compact
   :li. Item one.
```

In the example above, the first tag SCRIPT/VS finds is the :H1 tag. This tag, its attributes and its text will be completely processed before the line with the :OL tag is processed.

Once a tag has been found, SCRIPT/VS performs the following processing:

1. Finds the attributes for the current tag
2. Identifies the residual text associated with the tag
3. Figures out which APF to invoke to process the tag
4. Processes the instructions in the APF
5. Processes the residual text if the APF did not process it.

Identifying Attributes

When SCRIPT/VS finds a tag, it automatically checks the attribute rules for the tag to see if any attributes are allowed. The :OL tag was defined as follows:

```
     .aa ol dsmlistm (vat) dsmelist
```

In this case, value attributes (vat) are allowed. In the case of the :H1 tag, there are no attribute rules on the .AA [Associate APF] control word, so SCRIPT/VS uses the general rules established with the .GS [GML Services] RULES control word.

If the tag being processed is not allowed to have any attributes, SCRIPT/VS does not scan the rest of the line for attributes. It assumes everything following the tag is residual text.

If attributes are allowed, SCRIPT/VS scans the markup to find the attributes.

There are two different kinds of attributes that can be used—value attributes and labelled attributes. Value attributes are single words which are either specified or not specified. For example, COMPACT and BREAK on the :DL tag are value attributes. These attributes are passed directly into the APF for the tag in the &* symbol array.

Labelled attributes are specified using an attribute name, an equals sign and the value of the attribute. For example, PAGE=yes is a labelled attribute. Labelled attributes are processed by the APF for the tag using the .GS [GML Services] EXATT control word. During the scanning process each labelled attribute (the kind with the equals sign) is saved along with its value in a special attribute list.

SCRIPT/VS scans the line with the tag on it looking for attributes and residual text and continues to scan subsequent lines until it finds either a markup/content separator (.), another tag or text if value attributes are not allowed.[6] As soon as one of these is found, SCRIPT/VS stops looking for attributes.

In our example above, the line with the :H1 tag on it is scanned. The STITLE attribute and its value, "Introduction," will be saved in a special attribute list. Because there is no markup/content separator on this line, SCRIPT/VS will scan the next line also. The ID attribute and its value are also saved in the list. The period following the ID value is the markup/content separator and automatically ends the scan for attributes.

The :OL tag has a value attribute (COMPACT) on it. When SCRIPT/VS gets to this tag, the COMPACT attribute will be saved during the scan. Notice that there is no markup/content separator here. A separator is not necessary because there is no text associated with the :OL tag. It is followed directly by another tag, :LI, which ends the scan for attributes.

Identifying the Residual Text

Once SCRIPT/VS has processed the markup for a tag, it identifies the text associated with the tag. This is called the *residual text*. Not all of the tags in the starter set have residual text, but SCRIPT/VS checks for it anyway.

All of the text on the line from the end of the markup, up to but not including the next tag, is considered to be the residual text associated with the tag. In our example, the line that ends the markup for the :H1 tag has nothing else on it. In this case, all of the text on the next line is considered to be the residual text. "Introduction to the GML Starter Set" is the residual text for the :H1 tag. The :OL tag has no residual text because it is followed by another tag.

The end of the residual text is defined as the end of the input line or the occurrence of another tag. Since the residual text line is ended by the occurrence of another tag, it can not, by definition, contain any tags. For example,

```
    :h1.The :hp1.starter set:ehp1. Profile
```

The residual text for the :H1 tag is just the word "The." The scan for residual text was ended by the occurrence of the :HP1 tag. Similarly the residual text for the :HP1 tag is "starter set" and the residual text for the :EHP1 tag is "Profile." In the case of this last tag, the scan for residual text is ended by the end of the line, rather than the occurrence of another tag.

Since residual text is a very important concept, let's look at another example:

```
    :titlep
    :title.
    First Line
    of the Title
```

In this example, there is no residual text at all for the :TITLEP tag. "First Line" is the residual text for the :TITLE tag. Notice that the residual text for this tag includes only the first input line following the tag. There was no text on the same line with the tag so the residual text is the next input line.

Residual text is not important for all tags. For example, the first line of text after a paragraph tag has no special meaning as residual text. If the residual text has special meaning, it must all be

[6] If no markup/content separator is used and attributes are allowed on the tag, SCRIPT/VS may not be able to tell what is text and what are attributes. This is why the markup/content separator is an important part of the mark-up.

entered on one line, such as with the :H1 tag. In the starter set there are some tags which use their residual text. These are:

:ALINE
:AUTHOR
:DATE
:DDHD
:DOCNUM
:DT
:DTHD
:FIGCAP
:GT
:H0-6
:IH1-3
:I1-3
:TITLE

All other tags either don't have any residual text or don't use it.

Finding APFs

After checking for residual text, SCRIPT/VS checks to see if the tag has been associated with an APF. In our example, the DSMHEAD1 macro, which has been associated with the :H1 tag, is invoked because of the following .AA [Associate APF] control word:

```
.aa h1 DSMHEAD1
```

If SCRIPT/VS encounters a tag that has not yet been associated with an APF, it attempts to find a macro or control word with the same name as the tag. If there is one, it is processed. If neither a macro nor control word is found, the user gets an error message and the tag appears as text in the formatted output.

If SCRIPT/VS finds any value attributes when it scans the lines, they are passed as parameters to the APF and are available in the *local symbols* &*1, &*2, and so on. In the case of the :OL tag, the value attribute COMPACT is passed to the DSMOLIST APF which processes the :OL tag.

APF Processing

The APFs provide the specific formatting instructions for the particular text element or structure identified by the tag. The APFs are made up of SCRIPT/VS control words, text, and calls to other macros. They may do such things as cause a page eject, redefine the column definitions, start indenting, change fonts, and so on—whatever is required to process the element.

Residual Text: If the APF wants to use the residual text of the tag (as in the case of headings), it can use the .GS [GML Services] SCAN control word to get the text:

```
.gs scan @head
```

This control word will cause the residual text to be transferred into a symbol named &@head. The name of the symbol used on the .GS [GML Services] SCAN control word line can be anything you like—as long as it's a valid symbol name.

The APF must then process it completely, because the .GS [GML Services] SCAN control word causes SCRIPT/VS not to process the residual text.

On the other hand, if the APF just wants to look at the residual text, but still wants SCRIPT/VS to process it automatically, it can use a .GS [GML Services] COPY control word. If the text is not *scanned*, but is either ignored or *copied*, it will be processed automatically by SCRIPT/VS after the APF has finished.

For example, an APF might simply want to know if there is any residual text. In this case you would want to use the .GS [GML Services] COPY control word to get a copy of the residual text and check its length using the &L' symbol attribute:

```
    .gs copy @head
    .if &L'&@head eq 0 .th .....
    .el .....
```

If the length was zero you would now know that there wasn't any residual text and could proceed accordingly. If there was residual text, SCRIPT/VS will still process it automatically for you.

The automatic processing of residual text includes using literal mode (so that text is not misinterpreted as controls) and automatic handling of word continuation. If you need to process the residual text yourself in the APF, you will need to be concerned about such problems.

One special concern with word continuation is when the residual text scanning is ended by another GML tag. If the APF for the first tag uses .GS SCAN, there may be continuation problems because the first APF does not have enough information to cause correct word continuation in all possible situations. The first APF cannot determine whether the residual text scanning was ended by another GML tag, or the end of an input line. If the residual text scan was ended by another GML tag, an extra blank may be incorrectly added at the end of the first GML tag's residual text. Continuation after the second GML tag may be incorrect also. To avoid this situation do one of the following: Use .GS COPY if you only need to look at the residual text, allowing SCRIPT/VS to process the residual text. If .GS SCAN is necessary, cause breaks before and after the residual text. If breaks before and after the residual text are inappropriate, do not put any other GML tags on the same input line with your GML tag.

Attributes: The value attributes passed to the APF as parameters can be processed directly in the APF. They are available in the local symbols &*1, &*2 and so on. For example, if the COMPACT attribute is specified on the :OL tag, the &*1 symbol will contain the word "compact" when we get to the APF for the :OL tag. The attribute value is passed in exactly as it was specified. It is not folded to uppercase. If you had specified "Compact" the value of &*1 will be "Compact." If you had specified "COMPACT" the value of &*1 will be "COMPACT."

The labelled attributes (specified with an "=") are processed by APFs using the .GS [GML Services] EXATT control word. For example, the APF for the :H1 tag processes the ID attribute using:

```
    .gs exatt id as DSM@IDS
```

When this control word is executed, SCRIPT/VS checks the names of the attributes given on the control word line (in this case "id") against the special list of attributes that SCRIPT/VS created when it scanned the markup. If a match is found, the macro specified on the .GS [GML Services] EXATT control word is executed, and the value of the attribute (the right side of the equals sign) is passed to the macro in &*1. If no macro name is given on the .GS [GML Services] EXATT control word, the macro with the same name as the attribute is executed.

In our example, the DSM@IDS macro is invoked when the ID attribute is found in the markup. If we don't specify the macro name (DSM@IDS) on the control word line, then a macro named ID is invoked. If there is no ID macro, a message is issued.

In the starter set, the attribute macros are almost always specified by name on the .GS [GML Services] EXATT control word lines. This is because all of our macro names must start with DSM and this is never the same as the attribute name so it must be invoked by name on the .GS [GML Services] EXATT control word.[7]

[7] The exception to this rule is that several of the attribute macros in the starter set are defined by other macros in the library. Since they are dynamically defined they don't exist as macros in the library and therefore do not have to have names beginning with "DSM."

If the .GS [GML Services] EXATT control word line includes an attribute that was not in the markup, nothing happens. In other words, in our example, if no ID attribute was specified on the :H1 tag nothing happens when we get to the .GS [GML Services] EXATT control word.

Symbol Substitution and Residual Text Processing

Symbol substitution is performed on residual text before residual text is processed. This has implications when the symbol in the residual text is reset by processing in the APF for the tag this residual text is associated with. The following example illustrates this situation:

```
.se count = 0
.dm xxx on
.se count = &count + 1
.gs scan *line
&*line
.dm off
.*
:xxx.the count is &count
```

Symbol substitution is performed before the XXX APF is executed. During symbol substitution the symbol "&count" is resolved to the value "0". Therefore, after the GS SCAN is performed, the symbol "&*line" contains the value "the count is 0" instead of "the count is &count". To avoid this situation, do not put symbols in residual text for tags, whose APFs may reset that symbol.

Symbol Substitution and Tags

Do not put GML tags into symbols. Using GML tags in this manner may result in improper processing of the GML tag, for example: .SE tag2=:H1.

About the Starter Set

There are approximately 150 different macros in the starter set. In order to make it easier to recognize symbol names and APF names, several naming conventions have been used in writing the starter set.

APF Naming Conventions

All of the APFs stored in the macro library begin with DSM which indicates that they are part of the Document Composition Facility program product.

The fourth character in the APF name indicates what kind of APF it is.

- DSM# indicates a *service* or *initialization* macro that is only called by other macros, APFs, and the profile.
- DSM@ indicates a macro that processes an attribute.
- DSME indicates an APF for an end-tag.
- DSM followed by any letter other than E indicates an APF for a start tag.

Symbol Naming Conventions

Several different symbol naming conventions are used in the starter set. Most of the starter set symbols start with @. This is done to reduce the possibility that users will have symbols with the same names.

All of the literal constants (text) used in the starter set are put into symbols that start with &LL@. These symbols are defined by the DSM#SETS macro. For example, the DSM#SETS macro defines a symbol named &LL@ToC to have a value of "Table of Contents." This symbol is then used on the .TC [Table of Contents] control word in the APF for the :TOC tag to specify the heading to be put on the table of contents page.

The primary reason for this approach to handling text constants is to facilitate changing the constants.

The symbol names are made as meaningful as possible without making them unnecessarily long. For example, symbols that hold a value for white space to be skipped begin with "@sk" and indention values begin with "@in."

It is also generally true that "h" indicates headings, "f"—figures, "n"—footnotes, "l"—lists[8], and "i"—index. This is true, for example, in the &@sk@f and &@in@f symbols where the "f" represents figures.

The DSM@MAC@ Symbol

Symbols can also be stored individually in the macro library, There is one such symbol in the GML macro library—the &DSM@MAC@ symbol—that contains the name of the macro library. This symbol is used by the profile to verify that the correct macro library is available for processing.

General Service Macros

Some of the macros that begin with DSM# can be especially useful to you in writing your own APFs to extend the starter set. These macros are listed below and are described in detail in either "Starter Set Initialization" on page 19 or "Miscellaneous" on page 163.

[8] Occasionally we use "d" for lists instead of "l" to avoid duplicate symbol names.

APF	*Description*
DSM#CNTX	Invalid tags are mapped to this macro to produce a message that states that the tag is out of context. The tag name is included in the message. It is used, for example, as the mapping for the :ALINE tag when an address structure is not currently in progress.
DSM#DUPL	This macro ends the current page and prepares to start a new page. If duplexing is in effect (with SYSVAR 'D'), this macro will cause a page eject until it gets to an odd-numbered page. It is used for processing document sections (abstract, preface, index, and so on).
DSM#MSG	This macro issues all of the GML starter set messages. All messages are collected here to facilitate translation to other languages.
DSM#RSET	This macro checks to see if any structures such as figures, examples, footnotes, or lists are open. If any are found open, they are closed (ended) and a message is issued. This macro is used primarily by the heading APFs to perform housekeeping functions before starting a new heading.
DSM#SETS	This macro defines all of the literal text strings used by the starter set. They are collected here to facilitate translation into other languages. This macro also defines the &date and &time symbols.
DSM#STYL	This macro sets up the page layout to be one column, two column or offset style. It is used in the starter set to change the layout between the different document sections, such as body and back matter. It accepts "one," "two," or "off" as a parameter. If no parameter is given, the style system variable, &SYSVARS, is used to determine the desired page layout. The default page layout is one column if SYSVAR 'S' is not specified on the SCRIPT/VS command line.
DSM#SUPR	This macro produces superscripts. The parameters passed to the macro print as superscripts for the 1403 printer and the IBM 3800 Printing Subsystem Model. For page printers, superscripts are created by shifting the baseline up and printing the number in a smaller font. For terminals and other line printers, the parameters passed are printed enclosed in parentheses.
DSM#YESN	This macro analyzes attribute values that can be only "yes" or "no." In the starter set it is used by the cross-reference APFs and is further described in detail in "Cross-References" on page 147.

Special Techniques

Much of the starter set processing is provided by fairly complex manipulations of symbols. Rather than explain these techniques repeatedly for each of the APFs that uses them, a single detailed explanation is given here for some of the techniques.

Validating Keywords

Some of the attributes and SYSVARs accept only a limited number of values.[9] In these cases, we need to ensure that the value the user specified is valid. This involves checking the value against a list of valid values to see if we recognize it. Then we reset the value specifically so we know exactly what format it is in.

Let's take a look at how the SYSVAR 'D' variable is handled. This system variable indicates whether or not duplexing is to be done.

System variables are passed to SCRIPT/VS on the SCRIPT command using the SYSVAR option. The values given on the command for the various system variables are put into special system symbols named &SYSVARA, &SYSVARB etc. The starter set uses only C, D, H, P, R, S, T, W and X.

Most of these variables are analyzed in the DSM#SETV macro which is called from DSMPROF3 during the starter set initialization process. The purpose of the processing is to establish fixed values. We know that the value of the SYSVAR will be uppercase if it was specified on the command, because SCRIPT/VS translates it to uppercase. However, someone could set the SYSVARs in the profile, in which case we wouldn't know if they are in upper- or lowercase.

There may also be synonyms that need to be converted to a single value. We need to be able to count on these symbols being set to specific values so we can test them easily when we need to.

We're going to convert whatever comes in as the value for &SYSVARD to a lowercase "no" or "yes." If &SYSVARD is not set, we'll set it to "no" to indicate no duplexing. Accomplishing this requires three steps:

1. Checking the specified values against a list of valid ones
2. Seeing if we found it on the list and if not, setting the default
3. Setting or resetting the symbol to a recognizable value.

[9] For a description of what SYSVARs are used in the starter set see the *Document Composition Facility: GML Starter Set User's Guide*, SH20-9186 and the *Document Composition Facility: GML Starter Set Reference* , SH20-9187.

```
.se *a = index '-NO-YES-DUPLEX-SIMPLEX-' '-&U'&SYSVARD.'
.if &*a eq 0 .se *a 1
.se SYSVARD = substr 'no yes yes    no' &*a 3
```

Figure 3. Validating Keywords: The technique illustrated here is used to check a system variable (SYSVAR) value or an attribute value against a specified list of valid values.

Check the Attribute Value

The first thing to do is check the value of &SYSVARD. In the first line shown in Figure 3, we look up the uppercase value of &SYSVARD in a string made up of all the values we recognize. In this case, the only ones we recognize are "NO," "YES," "DUPLEX," and "SIMPLEX."

Each possibility is prefixed with a dash so that, for example, a value of "plex" is not recognized as valid. However, legitimate abbreviations such "y" for yes are recognized as valid.

We're using the uppercase attribute of the &SYSVARD symbol for comparison purposes and have prefixed it with a dash also.

Suppose that &SYSVARD was specified on the command as "No." Let's examine how this line will substitute:

```
.se *a = index '-NO-YES-DUPLEX-SIMPLEX-' '-&U'&SYSVARD.'
.se *a = index '-NO-YES-DUPLEX-SIMPLEX-' '-&U'No'
.se *a = index '-NO-YES-DUPLEX-SIMPLEX-' '-NO'
```

The right-hand string (or search argument) in the line above will resolve to "-NO" and &*a will end up set to 1 because "-NO" starts in the first position of the string we're looking in.

See if we Recognized the Value

The next thing to do is determine whether we recognized the value by testing the value of &*a. This is the second line in Figure 3.

If the uppercase value of &SYSVARD prefixed with a dash (-) was found in the string we looked in, &*a would be set to the starting position. (See the INDEX parameter of the .SE [Set Symbol] control word in the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for more details.)

If &*a is zero, it means that &SYSVARD was none of the things we thought it could be, so we will set &*a to 1. We'll see why later.

Reset SYSVAR to a Known Value

The next and last step is to get &SYSVARD set up the way we want it. It could have been entered in either upper- or lowercase and four different words are valid with "duplex" equivalent to "yes," and "simplex" equivalent to "no." In the third line in Figure 3, we reset &SYSVARD using the substring function of the .SE [Set Symbol] control word. The source string is set up to all "no"s and "yes"s in the same positions as we had the corresponding terms in the .SE [Set Symbol] INDEX control word line above.

```
line 1:    -NO-YES-DUPLEX-SIMPLEX
            |  |   |      |
            1  4   8      15
            |  |   |      |
line 3:     no yes yes    no
```

The &*a value is used as a starting point and we'll always take three characters.

The "1" we used above as the default value for &*a translates here to taking the substring starting at position 1 for three characters—in other words, "no."

Notice that we had to leave extra spaces between the second yes and the second no. This is necessary to make the two strings correspond exactly. If &*a is 4 after the index function because &SYSVARD was "Yes," then &SYSVARD will come out of the substring function set to "yes," because we'll take three characters from the string starting with the fourth character.

```
.se SYSVARD = substr 'no yes yes    no' &*a 3
.se SYSVARD = substr 'no yes yes    no' 4 3
.se SYSVARD = 'yes
```

Self-Modifying Macros

Several of the macros in the starter set modify themselves by redefining the entire macro, undefining the macro or changing one or more lines of the macro.

Reclaiming Space

When the macro is not going to be used again it is best to remove the definition from storage. The macros in the starter set that reclaim their storage do so in one of two ways:

- the macro is undefined

  ```
  .dm dsm#sets off
  ```

- the macro is redefined to a comment.

  ```
  .dm dsm#sets /.*
  ```

In either case, the macro's definition is deleted from storage. In the second method shown, the definition is replaced with a comment line. There is one significant difference between these two methods and that is what happens if the macro is called again. This could be on the second pass.

If the macro is in storage, SCRIPT/VS uses the copy that is in storage rather than the library copy. If the macro has been undefined (with .DM [Define Macro] macro OFF), this means that it is no longer in storage. In this case, the next time the macro is needed, SCRIPT/VS fetches it from the library. This way the real macro, as it exists in the library, is always used even if it defines itself to "off."

Alternatively, if the macro has been redefined to be a comment, SCRIPT/VS finds the macro definition (the comment line) in storage and processes it. The result is that the macro definition in the library is totally bypassed on subsequent calls which may or may not be okay depending on exactly what the macro does. If the macro only needs to be executed once during a formatting run using the approach of redefining the macro to a comment is fine.

For this reason, the first method described—undefining the macro—is the method used in the starter set. On the second pass, we want to use the real macro definition from the library again. The only macros that do this are the initialization macros called from DSMPROF3 during initialization.

Replacing a Line of a Macro

Another example of macros which modify themselves involves a line of the macro replacing itself with another line. The macros that do this, such as DSM#XLST and DSM#WRIT, do so to make sure that the macro is executed only once. Both of these macros may be called either from the :EGDOC tag or from the epifile section of DSMPROF3. They redefine themselves to make sure that they are not called by both :EGDOC and the epifile.

In both cases the first line of the macro is replaced with a new control word line which contains a .ME [Macro Exit] control word. This means that the first time the macro is executed the first line is changed to a .ME [Macro Exit] control word and the second time the macro is executed the .ME [Macro Exit] control word is the first line encountered and causes the macro to end immediately.

For example, the first line of the DSM#XLST macro looks like this:

```
.dm DSM#XLST(&$LNUM.) /.me
```

The &$LNUM symbol resolves to the current macro line number which in this case is 10 because this is the first line of the DSM#XLST macro. This causes the first line of the macro to be redefined to the .ME [Macro Exit] control word.[10] The next time the macro is executed it will end immediately without further processing.

Removing a Line of a Macro

Another technique that is used within the starter set involves a macro modifying itself by removing a line of the macro. This technique is used in the cross-reference listing macros (DSM#XRFD, DSM#XRFN, DSM#XRFH, DSM#XRFI, and DSM#XRFN). These macros are called repeatedly to produce one line of the cross reference listing at a time. The first time each of the macros is called it needs to call the DSM#SETX macro to generate a heading for the id section such as "Heading IDs" or "Figure IDs." Since this heading is produced only the first time the macro is called and is not needed on subsequent calls, these macros redefine themselves to remove the call to DSM#SETX after the first time. It looks like this:

```
.dm DSM#XRFH(&$LNUM.) off &$CW..DSM#SETX  H
```

The &$LNUM symbol resolves to the current macro line number. The line is constructed in two parts separated by a control word separator (&$CW). The first time the line is processed, the entire line is picked up and split into its two parts. Part one of the line is processed first and it removes the original line from the macro by setting it "OFF." The second part of the line is then processed and results in the DSM#SETX macro being called with "H" as a parameter.

The next time the DSM#XRFH macro is executed, the line is not present at all and the DSM#SETX macro is not called.

Another APF that uses this technique is the DSMLISTM APF. The first time DSMLISTM is called, it removes its first three lines and defines three single-line macros.

```
.dm dsm#listm(&$LNUM.) off &$CW..dm @termhi /.se @hi@l = &*1/
.dm dsm#listm(&$LNUM.) off &$CW..dm @tsize /.se @in@l '&*1/
.dm dsm#listm(&$LNUM.) off &$CW..dm @headhi /.se @hi@hd '&*1/
```

[10] Redefining the macro to end immediately is no different really than redefining the macro to be a comment except that the macro is not removed from storage. The technique of redefining lines of macro is used for various purposes, not just avoiding re-executing the macro.

Each line defines a simple macro that is used to process an attribute of the list tags. Because these macro definitions need be done only once—the first time we call the DSMLISTM macro—the lines redefine themselves to off.

Setting Caller's Local Symbol

There are several places in the starter set macros where a macro ends with a .ME [Macro Exit] control word line that also has a .SE [Set Symbol] control word on the same line, as in:

```
      .me .se *id '&*
```

The .ME [Macro Exit] control word ends the current macro. The .SE [Set Symbol] control word line is executed after the macro has ended and will set a local symbol for the calling macro. In the example shown above, the &*id local symbol belongs to the *calling* macro, not the macro that is being ended. The &* represents the parameters that were passed as a local symbol into the macro which is ending.

For example, the :HDREF tag is processed by the DSMHDREF APF. This APF calls the DSM@RFID macro to process the REFID attribute on the :HDREF tag. The value of the REFID attribute is passed to DSM@RFID in the &* symbol. This value is not available to the DSMHDREF macro—only to the DSM@RFID macro. The DSM@RFID macro has just one line in it:

```
     .me .se *id '&*
```

The DSM@RFID macro ends immediately and stacks the .SE [Set Symbol] control word line to be executed back in the DSMHDREF APF. The &*id symbol is local to the DSMHDREF macro, not the DSM@RFID macro. The &* symbol, however, belongs to the DSM@RFID ma ro. In order to work correctly we want the &* symbol to resolve to the value it has for the DSM@RFID macro. Since symbol substitution occurs before control word processing, the &* symbol is resolved before the .ME [Macro Exit] control word ends the DSM@RFID macro which is what we want to happen.

This technique is used because we make every attempt possible to use local symbols rather than global symbols in order to avoid creating and maintaining unnecessary symbols. Many symbol values only have meaning for the duration of the macro that is creating them.

Saving and Restoring Environments

The starter set frequently saves and restores the formatting environment using the .SA [Save Status] and .RE [Restore Status] control words. The .SA [Save Status] control word saves a copy of the current values of the active and page environments and the translate tables. The .RE [Restore Status] control word restores the saved values of the active and page environments and the translate tables.

The active environment contains formatting control values including such things as column layout, baseline position, font save stack, centering, indention control, uppercase and underscoring, and formatting mode (.FO). The page environment includes page dimension values such as page length, line length, column line length and page margins. The translate tables that are saved and restored includes the input translation table (.TI) and the output translation table (.TR). The contents of these environments are given in the *Document Composition Facility: SCRIPT/VS Language Reference.*

The formatting environment (which includes all three of these areas) is also saved and restored by keeps, floats, footnotes and named areas.

The starter set uses .SA [Save Status] and .RE [Restore Status] primarily in situations where many changes are going to be made to the formatting environment. In these cases, where things such as formatting mode and indention are going to be changed, it is far easier to simply restore the previous environment than to reset each value with the appropriate control word.

Setting Symbols

Occasionally in the starter set it is necessary to set a number of symbols all at once. This could be accomplished using a lot of .SE [Set Symbol] control words. However, there is a quicker way to accomplish the same thing using the .GS [GML Services] ARGS and VARS control words. For example, in DSMPROF3 the following line:

```
.gs args 123456     123        top       page
.gs var  @olistnest @ulistnest @figplace @figwdith
```

is totally equivalent to

```
.se @olistnest '123456
.se @ulistnest '123
.se @figplace 'top
.se @figwidth 'page
```

The primary reason for using the .GS [GML Services] control word instead of a number of .SE [Set Symbol] control words is performance. In two lines we can set many symbols rather than just two symbols. This technique can not be used for quickly constructing arrays.

Note: Using the .GS [GML Services] ARGS control word destroys the &* array.

Enforcing Structure

There are several different techniques that are used in the starter set to enforce particular structure among GML tags. For example, :FIGCAP and :FIGDESC tags are not allowed outside of a figure because they would have no meaning. This relationship between the tags is enforced through mapping and re-mapping the tags. The :FIGCAP and :FIGDESC tags are mapped to a special macro named DSM#CNTX in the profile.[11] If these tags are used outside of a figure, the user will get a message that the tags are out of context. When the :FIG tag is processed it re-maps :FIGCAP and :FIGDESC to the DSMFCAP and DSMFDESC macros, respectively. When the :EFIG tag is processed, :FIGCAP and :FIGDESC are once more mapped to DSM#CNTX because they are no longer valid tags.

A second technique is used to enforce some of the other restrictions which are built into the starter set. For example, footnotes are not allowed within figures or examples. This restriction results from the fact that SCRIPT/VS does not allow footnotes to be processed within floats or keeps. Therefore, the starter set keeps track of when a figure, example or footnote is currently being processed and disallows the others from starting. This is done using the &@state symbol. When none of the conflicting structures are in progress, &@state has a value of "open." When an example is started, &@state is set to "Exmpl." Before starting a footnote, figure or example the value of &@state is checked to make sure it is "open." If it isn't, a warning message is issued and the current tag is ignored. This avoids having the user get a message from SCRIPT/VS about keeps, floats or footnotes. The starter set can avoid creating the situation and can issue a more meaningful message that includes which tags caused the problem.

In the case of keeps, floats and footnotes, the relevant information is also available in the &$ENV symbol. This symbol could be checked to see if a keep or float is in progress before starting the footnote, and so on. However, &@state is also used to indicate when a title page is being processed so we would need a special symbol anyway.

[11] See "Miscellaneous" on page 163 for details on the DSM#CNTX macro.

Starter Set Initialization

Overview

When SCRIPT/VS is run with the DSMPROF3 profile a set of macros is invoked initially to set up the appropriate formatting environment for the starter set. The initialization performed by the profile and its related macros includes:

- Mapping tags
- Defining fonts
- Defining running headings, footings, and heading levels
- Initializing many symbols that will be used in the various APFs and macros, and
- Defining the page layout.

We have attempted to put as much formatting control into the profile as possible. There are a large number of symbols which are set in DSMPROF3 which control the formatting that will take place in the APFs. However, not everything can be standardized for the entire document. Some things need to function off of the column layout style or the document section. Therefore, some layout control exists in macros rather than in the profile. The figure below shows the macros and the sequence in which they are called during initialization.

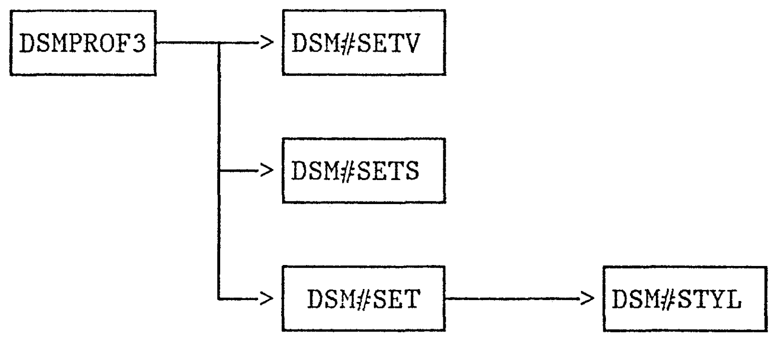


Figure 4. **Starter Set Initialization Macros:** This diagram shows the calling sequence during initialization of the starter set.

- The DSM#SETV macro processes the SYSVARs specified on the SCRIPT command.
- The DSM#SETS macro defines literal constant symbols for use in the starter set and the &date and &time symbols.
- The DSM#SET macro initializes various counters and miscellaneous symbols used in the starter set.
- The DSM#STYL macro sets up the page layout. It is also called by the various document section macros (DSMFRONT, DSMBODY, DSMAPPD, DSMBACKM and DSMINDEX).

The DSMPROF3 Profile

DSMPROF3 is the profile for DCF Release 3 GML general documents. It must be specified on the command line or be imbedded at the beginning of the primary input file in order to enable starter set processing. DSMPROF3 performs the following functions:

1. Determines whether Release 3 of the Document Composition Facility is being used. If not, a severe error message is issued.
2. Retrieves the macro library name from the DSM@MAC@ symbol in the library. If the maclib is not DSMGML3, a severe error message is issued and processing ends.
3. Defines the indention and skip amounts for lists, paragraphs, footnotes, figures, long quotations and examples. The symbols for indention are named &@in@x and the skip symbols are named &@sk@x where "x" stands for some unique letter that is used to indicate each particular text element. For example, for a figure the indention symbol is &@in@f and the skip symbol is &@sk@f. See Figure 5 on page 21 for a complete listing of the indention and skip symbol settings. The .GS [GML Services] ARGS and VARS control word are used as a quick, efficient method for setting many symbols all at once. See "Special Techniques" on page 13 for more details on this technique of setting symbols.

Description	Letter	Indent	Skip
Definition Lists	(d)	10	1
Ordered Lists	(o)	4	.75
Unordered Lists	(u)	4	.75
Simple Lists	(s)	4	.75
Glossary Lists	(g)	0	.75
Paragraphs	(p)	0	.75
Footnotes	(n)	none	1
Long Quotes	(q)	3	1
Undefined Lists	(z)	4	.75
Examples	(x)	2	1
Figures	(f)	2	1

Figure 5. Indention and Skip Initialization

Note: Several of the skip values are set to .75. For page printers, this results in three-quarters of a line space being used. For line printers, such as the 1403 or the 3800 Printing Subsystem Model 1, this is rounded up to one line space.

4. Sets the default highlight fonts for lists. The default for definition terms (&@hi@d) and glossary terms (&@hi@g) is highlight level 2. The default for definition list headings (&@hi@h) is highlight level 3.

5. Sets the skip value for before and after lists (&@sk@ls) to .75.

6. Defines &@olistnest and &@ulistnest to indicate the sequence of identifiers to be used in ordered and unordered lists for list items. The identifier definitions are described below. The list item identifiers can be changed by either changing the number sequence in &@olistnest or &@ulistnest or by changing the definitions of the identifiers. Controlling the identifiers is discussed in "Modifications to List Processing" on page 108.

7. Defines the default figure placement and width (&@figplace and &@figwidth) to be "top" and "page" respectively. These are used by the DSMFIG macro when processing the :FIG tag.

8. Turns macro substitution on. This is because the APFs for the starter set are macros. Without macro substitution on, SCRIPT/VS could not process any of the tags.

9. Enables use of the library for macros but *not* for symbols. Enabling the library for symbols is expensive in terms of performance and serves no purpose because only one of the starter set symbols is in the library.[12]

10. Defines the GML delimiters as ":" for start tags and ":e" for end tags.

11. Turns on scanning for GML tags with the .GS [GML Services] TAG control word.

12. Defines the continuation character to be hexadecimal "03."

13. Sets up the default GML attribute scanning rules (att novat stop and nomsg) for tags and (noatt) for end tags. These scanning rules will be used for all tags unless over-ridden on the .AA [Associate APF] control word line for the tag. If a particular attribute rule has been defined in the profile using .GS [GML Services] RULES control word and a contradictory rule is also specified with a .AA [Associate APF] control word, the .AA [Associate APF] control word rule is used.

[12] The DSM@MAC@ symbol is explicitly defined as being in the library at the beginning of the profile.

14. Defines the list item identifiers for ordered and unordered lists using the .DV [Define Variable] control word. Symbol substitution is turned off for the definition of the ordered list identifiers to prevent the symbol attributes, such as &a', from resolving during the definition of the identifiers. We want to save these symbol attributes and resolve them when we go to use the identifier on a list item.

    The names of the defined variables are constructed using "@id@l@" followed by the type of list (u or o) and the list nesting level the identifier is to be used for. For example,

    ```
    .dv @id@l@u1 /&X'9f
    ```

    defines the identifier for the first level of unordered list to be a hexadecimal "9F" which is a bullet in the page printer fonts. Similarly,

    ```
    .dv @id@l@u2 font @pi@u1 /&X'db
    ```

    defines the second level unordered list identifier to be a hexadecimal "db" in the font named @pi@ul. See the description of the .DV [Define Variable] control word in the *Document Composition Facility: SCRIPT/VS Language Reference* for additional details.

    Several sets of unordered list item identifiers are defined: one set if the output is going to a printer (SYSOUT = PRINT), one set if the output is not going to a printer, and then a special set for page printers (SYSOUT = PAGE). For list item identifiers for page printers, the Pi font (@pi@ul) is defined. For the IBM 3820 Page Printer and 3800 Printing Subsystem Model 3, the Pi font definition is slightly different because the typeface name and the codepage name for the Pi font are slightly different. The @pi@ul font is used to obtain various special characters for second and subsequent levels of unordered lists. Since there are so many special characters available for the 4250 printer, IBM 3820 Page Printer, and 3800 Printing Subsystem Model 3 in the Pi font, the &@ulistnest symbol is redefined to use five separate levels of identifiers before reusing the first again.

    For unordered lists, five levels of identifiers are defined for all output types but only the first three are specified (and therefore used) in &@ulistnest. Nine levels of identifiers are defined for ordered lists but only 6 are used.[13]

    A single identifier (*) is defined for *undefined* list types (z). The list type z is not documented. To produce a list where all the items are identified with an asterisk you can use the :L tag.

    ```
    :l
    ```

    Examples on changing list item identifiers are given in "Modifications to List Processing" on page 108.

15. Defines symbols for the GML tag delimiter, the ampersand, and the semicolon. These are the &gml, &amp, and &semi symbols respectively. The .DV [Define Variable] control word is used to assign these symbols to the correct output character.

16. Uses the .DR [Define Rule] control word to define several rules for use in the starter set. The first one is for figure rules. It has no parameters on it because we want to use the default rule which is .3mm. The rule definition is provided to make it easy for users to change the rule.

    The second rule is for the footnote leader. A weight is given that is slightly lighter (thinner) than the default rule—.2mm. The length of the footnote leader is also controlled from the profile. See item 21. on page 23.

[13] The reason we define more identifiers than we use is simply to facilitate user modifications to the list item identifiers. It is a simple matter to change around which of the defined identifiers are actually used. See "Modifications to List Processing" on page 108 for examples of how to do this.

17. Contains a .SE [Set Symbol] control word line that is commented out. It sets the &@bodyhead1 symbol to "Chapter." This is provided to facilitate creating a prefix for level one headings in the body of the document. If the comment (.*) is removed, headings will be labelled "Chapter 1 ...," "Chapter 2 ...," and so on. The prefixing and numbering is handled by the DSMHEAD1 APF described in "Headings" on page 75.

18. Sets line spacing controls to permit skips, spaces and text lines to be increased by a factor of 1.1 or decreased by a factor of .9 when necessary for vertical justification.

19. Turns on vertical justification.

20. Turns on hyphenation. Specifies that the default dictionary will be used but not the algorithmic hyphenator. The range parameter on the .HY [Hyphenate] control word provides the compression and expansion ranges. The ladder parameter specifies the number of consecutive lines that can be hyphenated. The MINPT, MAXPT and MINWORD values control when and where hyphenation points can occur.

21. Defines several spacing values for footnotes in the profile to provide easy modifications. The width of the gutter between the columns for two-column format is set in &@gutter to 4 spaces. The length of the footnote leader is set in &@fnldrlen to 16 spaces. A line spacing value, &@ttllo, is also defined as 1.2 (or 120%) for use on the title page. This factor is used to increase the line spacing for multiple title lines on page printers to provide greater visual separation between the lines.

22. Calls DSM#SETV to process the SYSVARs specified on the command line.

23. Calls DSM#SETS to define symbols for various literal strings (words) that are used throughout the starter set. These literal strings are put into symbols and the symbol definitions are collected into a single macro to facilitate changing them.

24. Defines spacing values for all headings.

    The &@hspbf symbol is used for the SPBF parameter on the .DH [Define Head Level] control word for head level zero and one. The value of &@hspbf is 0 for line printers and 1.3 inches for page printers.

    Since skips before a heading are thrown away when they are on the top of a page, we used spaces before the heading to "sink" the heading a little. This was done for stylistic reasons to set the level zero and one headings off a little from other headings. To remove this space simply set the value of &@hspbf to zero.

    The &@h0sp symbol is set to 5 lines and is used below for the SPAF parameter on the .DH [Define Head Level] control word for head level zero.

    The &@h1sp symbol is set to 3 lines and is used below for the SPAF parameter on the .DH [Define Head Level] control word for head level one. This value is modified below if we are formatting for a page printer.

    For head levels two through four in Release 2 the amount of space around the headings was a function of the style (SYSVAR 'S') of the document. Less space was provided around the headings in offset style. Now, in Release 3, the spacing around these headings is set by device. The initial values are set here in the profile to be 3 skips before and 2 spaces after for head 2 through 4. These values are put into the &@h2sk, &@h2sp, &@h3sk, &@h3sp, &@h4sk and &@h4sp symbols. We alter these values further if it turns out that we are formatting for a page printer.

25. Defines all of the fonts used in the starter set except for the Pi font used for unordered lists for page printers. Each output device has its own particular characteristics and font capabilities. Therefore, each has its own set of font definitions.

The following device specific functions are performed:

a. Defines highlight level 0 to be the body or default font (&$CHAR(1)).

b. Selects the style of superscript numbers and puts them into a symbol (&@suprstyl). The three styles or formats available are:

   - parentheses (&@suprstyl = parens)
   - true superscript numbers (&@suprstyl = nums)
   - superscripts created by shifting the baseline up and using a smaller font (&@suprstyl = shifts)

   The "nums" style is used for the 1403 and the IBM 3800 Printing Subsystem Model 1. The baseline shift method is used for all page printers.

c. Defines a set of highlight, heading, and table of contents fonts for each combination of output device (1403, 3800, and 2741) and the number of fonts given on the CHARS option. These are shown in Figure 6 on page 25. For page printers, many more fonts have been selected because of the greater range and number of fonts available for these devices.

   Note: The font definitions for the IBM 3820 Page Printer and 3800 Printing Subsystem Model 3 are identical to those for the 4250 printer except for the example font (xmpfont).

   In many places in the starter set, font changes are performed only for page printers by using the .BF [Begin Font] control word list capability. For example,

   ```
   .bf fnt =
   ```

   is used in footnotes where a font change is desired for the page printers only. For line devices, "fnt" is an undefined font. The "=" on the .BF [Begin Font] control word line causes the current font to be restarted. This eliminates the need for defining a "fnt" font for each possible device.

   In the case of heading and table of contents entries, it is not as simple. The .DH [Define Head Level] control word accepts font names for headings and table of contents entries. However, it does not have the same capability to accept a list of fonts to use. Therefore, it was necessary to define the fonts for headings and table of contents entries for all possible devices. See "Headings" on page 75 for more details about headings and table of contents processing.

   Alternate highlight fonts are defined for page printers. This is done because the highlight fonts are *type* defined fonts—that is, they simply change the current font into its italic or bold version. It is possible to get into situations where there is no italic or bold equivalent for page printers, such as in examples (:XMP) where the font used is Prestige or Prestige Elite. There may be no italic prestige font available for use. In these cases, alternative font definitions of underscore (althi1), uppercase (althi2) and both (althi3) are used for highlight fonts. There is no alternate highlight zero for "hi0" because this font is either &$CHAR(1) for line devices or is defined as type normal for page printers.[14]

   The font definitions for page printers assume that the default body font is a 10-point font. The typeface is not important to these definitions and is never specified. The only two typefaces specified are for examples and for unordered list item identifiers.

[14] It is possible to get into a situation where there is no "normal" font which will result in an error message if :HP0 is used.

FONT NAME	1403 2741	3270 3800(1)	3800(2)	3800(3)	3800(4)	3800-3/ 3820/4250
HI1	US	US	CHAR(1) US	CHAR(2)	CHAR(2)	italic
HI2	OS	UP	CHAR(2)	CHAR(3)	CHAR(3)	bold
HI3	OS US	UP US	CHAR(2) US	CHAR(3) US	CHAR(4)	bold italic
HD0	OS UP US	UP US	CHAR(2) UP US	CHAR(3) UP US	CHAR(4) UP	bold 20 italic
HD1	OS UP US	UP US	CHAR(2) UP US	CHAR(3) UP US	CHAR(4) UP	bold 20
HD2	OS US UP	UP US	CHAR(2) UP US	CHAR(3) UP US	CHAR(4) UP	bold 18 italic
HD3	OS US	UP US	CHAR(2) US	CHAR(3) US	CHAR(4)	bold 14
HD4	OS	UP	CHAR(2)	CHAR(3)	CHAR(3)	bold 12 italic
HD5	OS	UP	CHAR(2)	CHAR(3)	CHAR(3)	bold
HD6	US	US	CHAR(1) US	CHAR(2)	CHAR(2)	bold italic
HD0TOC	OS	UP	CHAR(2)	CHAR(3)	CHAR(3)	bold 10 UP
HD1TOC	OS	UP	CHAR(2)	CHAR(3)	CHAR(3)	bold 10
HD2TOC	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)	10
HD3TOC	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)	CHAR(1)	10

Figure 6. Heading, Highlight and Table of Contents Font Definitions

Key: OS = overstrike, US = underscore, UP = uppercase. The page printer (3800-3/3820/4250) font definitions such as "bold 20 italic" indicate the bold italic version of the body font in a pointsize of 20 points.

26. Calls DSM#SET to define various symbols for use in the starter set. The DSM#SET macro is described in detail below.

27. Defines two symbols, &@oquote and &@cquote, to provide the appropriate open and close quotation marks for each level of nested inline quotations. For page printers, typographical style quotation mark character are used. These symbols are used by the APFs for the :Q and :EQ tags.

    Not all uses for quotation marks are handled by the :Q tags. It is often necessary to use just single quotation marks. While it is possible to type these quotations marks directly, this will not always produce desirable results on page printers. Therefore, four additional symbols are defined to generate quotations marks directly in text. These are

    - &oqq - double opening quotes
    - &oq - single opening quotes
    - &cqq - double closing quotes
    - &cq - single closing quotes

These are defined separately for line printers and for page printers because different characters must be used.

28. Defines heading characteristics and control symbols:

    a. If the &@bodyhead1 .SE [Set Symbol] control word line described above is not commented out, the &@head1 symbol is used to contain a special prefix for level one headings. The &@head1 symbol is set "off" initially because only headings in the body and the appendix can be prefixed. The &@head1 symbol will be set to the value of &@bodyhead1 by the DSMBODY APF when the :BODY tag is processed.

    b. All headings are defined with no hyphenation permitted (NOHY).

    c. Each heading level has its own font. Heading levels zero through three also have a font specified for table of contents entries.

    d. On the .DH [Define Head Level] control words we need to specify whether or not the headings are to be numbered. This depends on the value of &SYSVARH. If &SYSVARH is not "no," it means that headings are going to be numbered so we set a local symbol, &*n, to "num." We will use this local symbol in the .DH [Define Head Level] control word lines. Additionally if we are going to number headings &SYSVARH will contain the starting number for the first level one heading. We use this value to initialize the heading counter with the .GS [GML Services] HCTR control word.

    e. The headings are all defined using the .DH [Define Head Level] control word. For the level zero and one headings there are two separate .DH [Define Head Level] control words simply because we couldn't fit all of the parameters for these headings on a single line.

    More detailed information and explanation about heading processing is given in "Headings" on page 75.

29. Saves and then restores the line length value to the default setting. This is done so we can tell if the user somehow got there first and changed the line length. If the default value is not the same as the saved value, we know the user has reset it. If not, we set line length to 6.8i for two-column and offset style formats. For one-column style the line length remains set to the default value.

30. Calls the DSM#STYL macro to establish the appropriate column format as specified with SYSVAR 'S'. This macro is described in detail below.

31. Defines the running heading and footing.

    The ***running heading*** centers the security classification. The .CE [Center] control word has a control word modifier on it because the &@sec symbol could start with a period or could contain semicolons (;) both of which could be misinterpreted without the control word modifier.

    The running heading is formatted in highlight level 2 for line devices and in the "@rh" font for page devices.

    Some space is left at the bottom of the running heading to provide some separation between the running heading and the body of the page.

    There are three ***running footing*** definitions. The running footing is formatted in the normal body font or in the "@rf" font for page devices. Space is left at the top of the running footing to provide separation from the body of the page.

    The first footing definition, which is used if duplexing is not in effect (&SYSVARD is "no"), formats the short heading (&@shead) on the left side of the page and the page number on the right side. The &@shead symbol contains either

a. The title of the document,
b. The short title of the document,
c. The last level zero or one heading, or
d. The last level zero or one short title value,
whichever was encountered most recently.

The other two running footing definitions are used if duplexing is in effect. In this case, a separate footing is defined for even pages and another for odd pages. The footing for odd pages formats the short title for the heading (&@shead) and the page number on the right margin. The even page footing formats the short title of the document (&@stitle) and the page number on the left margin of the page. See the running footings in this book as an example of the even and odd running footings used when preparing output for duplexing.

32. Maps the GML starter set tags to the appropriate APFs. Tags that are not valid initially because they required some text structure to be started are mapped to the DSM#CNTX APF. This APF, which is discussed in detail in "Miscellaneous" on page 163, issues a message that the tag is "out of context."

Tag	APF	Rules	End APF
ABSTRACT	DSMABSTR	noatt	
ADDRESS	DSMADDR	noatt	DSMEADDR
ALINE	DSM#CNTX	noatt	
APPENDIX	DSMAPPD	noatt	
AUTHOR	DSM#CNTX	noatt	
BACKM	DSMBACKM	noatt	
BODY	DSMBODY	noatt	
CIT	DSMCIT	noatt	DSMECIT
DATE	DSM#CNTX	noatt	
DD	DSM#CNTX	noatt	
DDHD	DSM#CNTX	noatt	
DL	DSMDLIST	vat	DSMELIST
DOCNUM	DSM#CNTX	noatt	
DT	DSM#CNTX	noatt	
DTHD	DSM#CNTX	noatt	
FIG	DSMFIG		DSMEFIG
FIGCAP	DSM#CNTX	noatt	
FIGDESC	DSM#CNTX	noatt	
FIGLIST	DSMFLIST	noatt	
FIGREF	DSMFGREF		
FN	DSMFTNT		DSMEFTNT
FNREF	DSMFNREF		
FRONTM	DSMFRONT	noatt	
GD	DSM#CNTX	noatt	
GDOC	DSMGDOC		DSMEGDOC
GL	DSMGLIST	vat	DSMELIST

Figure 7. Initial Mapping for GML Tags (Part 1 of 2)

Tag	APF	Rules	End APF
GT	DSM#CNTX	noatt	
HDREF	DSMHDREF		
HP	BF	noatt	DSMEHP
HP0	DSMHP0	noatt	DSMEHP
HP1	DSMHP1	noatt	DSMEHP
HP2	DSMHP2	noatt	DSMEHP
HP3	DSMHP3	noatt	DSMEHP
H0	DSMHEAD0		
H1	DSMHEAD1		
H2	DSMHEAD2		
H3	DSMHEAD3		
H4	DSMHEAD4		
H5	DSMHEAD5		
H6	DSMHEAD6		
INDEX	DSMINDEX	noatt	
L	DSMLISTM	vat	DSMELIST
LI	DSM#CNTX		
LIREF	DSMLIREF		
LP	DSM#CNTX	noatt	
LQ	DSMLQUOT	noatt	DSMELQU
NOTE	DSMNOTE	noatt	
OL	DSMOLIST	vat	DSMELIST
P	DSMPARA	noatt	
PC	DSMPCONT	noatt	
PREFACE	DSMPREF	noatt	
PSC	DSMPSC		DSMEPSC
Q	DSMQUOTE	noatt	DSMEQUOT
SL	DSMSLIST	vat	DSMELIST
TITLE	DSM#CNTX		
TITLEP	DSMTTLEP	noatt	DSMETTLP
TOC	DSMTOC	noatt	
UL	DSMULIST	vat	DSMELIST
XMP	DSMXMP		DSMEXMP

Figure 7. Initial Mapping for GML Tags (Part 2 of 2)

33. Maps the index tags. If no indexing is to be done the index tags are mapped to a *dummy* APF that simply removes the tag and its residual text from the document. If indexing is being done, the tags are mapped to their real APFs.

    The .AA [Associate APF] control word lines for the index tags are constructed using symbols whose values are determined by whether or not indexing has been requested. The .GS [GML Services] ARGS control word defines the symbols used on the .AA [Associate APF] control word line.

When indexing has been requested the following .GS ARGs line is used:

```
.gs args 1   2   3   dsmindx  dsmihd   dsmiref
```

which is the same as setting:

```
.se *1 = 1
.se *2 = 2
.se *3 = 3
.se *4 = dsmindx
.se *5 = dsmihd
.se *6 = dsmiref
```

When indexing has not been requested this .GS ARGs line is used:

```
.gs args              ' ' ' ' ' ' dsmidmmy dsmidmmy null
```

which the same as setting:

```
.se *1 = ' '
.se *2 = ' '
.se *3 = ' '
.se *4 = dsmidmmy
.se *5 = dsmidmmy
.se *6 = null
```

When the compound symbols on the .AA [Associate APF] control word lines are processed the APF names are resolves as follows:

```
.aa il  &*4.&*1
.aa il  &*4.1
.aa il  dsmindx1
```

when indexing has been requested. When indexing has not been requested, this same line resolves as follows:

```
.aa il  &*4.&*1
.aa il  &*4.
.aa il  dsmiddmy
```

Figure 8 shows the resulting APF names used for both indexing and not indexing.

Tag	Indexing	Not Indexing
I1	DSMINDX1	DSMIDMMY
I2	DSMINDX2	DSMIDMMY
I3	DSMINDX3	DSMIDMMY
IH1	DSMIHD1	DSMIDMMY
IH2	DSMIHD2	DSMIDMMY
IH3	DSMIHD3	DSMIDMMY
IREF	DSMIREF	null

Figure 8. Index Tag Mapping

34. Resets the .GS [GML Services] ARGS control word arguments to null to undefine the symbols &*1, &*2 and so on. If we didn't so this these symbols would still contain the last values we had set them to because local symbols used in a file are not true local symbols in the sense of becoming undefined at the end of the file.

35. The profile ends. What follows the .EF [End of File] control word is the epifile.

36. *The Epifile.* This part of DSMPROF3 will be automatically invoked by SCRIPT/VS at the end of all processing and is used to create the cross reference listing, the imbed trace and the SYSVAR 'W' file.

    If the profile is not specified on the command but rather is imbedded by some user profile, the epifile described here may never get processed. In this case the epifile of the *user* profile will be processed rather than the epifile of DSMPROF3. To get the DSMPROF3 epifile processed, the user profile needs to imbed DSMPROF3 in its epifile.

37. Calls the DSM#WRIT macro to generate the SYSVAR 'W' file of IDs if SYSVAR 'W' was specified and this is the last pass (&@lastpass is "yes").

38. Calls the DSM#XLST macro to generate the cross reference listing and imbed trace if cross referencing has been requested (&SYSVARX is "yes") and this is the last pass (&@lastpass is "yes").

Initialization Macros

Several macros and the DSM@MAC@ symbol are used during the initialization process from DSMPROF3. These are described below in the order in which they are used.

DSM@MAC@

The DSM@MAC@ symbol is set to "DSMGML3" and is used by DSMPROF3 to verify that the correct macro library is available. This is a *symbol* stored in the macro library, not a macro.

DSM#SETV

The DSM#SETV macro processes most of the the system variables (SYSVARs) specified on the SCRIPT command, resets them to standardized values for easy testing, and sets up defaults for them if they were not specified on the command.[15]

The technique used in this macro to validate the SYSVAR values and reset them is detailed in "Validating Keywords" on page 13 and therefore is not explained here. The SYSVARs processed by DSM#SETV are:

1. SYSVAR 'D' controls whether or not the document is to be formatted for duplexing. This macro sets &SYSVARD to either "no" or "yes" based on the value given on the command. The default is no duplexing.

2. SYSVAR 'H' controls whether or not heading levels 0 through 4 are numbered in the body of the document. The macro sets &SYSVARH to "no," "1.0" or the value given on the command. The default is no numbers for headings.

3. SYSVAR 'P' sets up a value to control the inclusion or exclusion of text and controls in a conditional section. This macro sets &SYSVARP to a null string if not given on the command.

[15] The one SYSVAR which is not processed by this macro is SYSVAR 'W' which is processed at the end of the formatting run by the DSM#WRIT macro.

4. SYSVAR 'R' specifies a file of cross reference IDs that is imbedded at the beginning of the formatting run to help resolve forward references more correctly.

   For the CMS and TSO environments, the DSMUTREF file is defined to be the file specified with SYSVAR 'R'. This file is then imbedded. The SYSVAR 'R' function is available only in CMS and TSO. The name of the file varies, depending on the environment. See "Cross-References" on page 147 for more details on the contents of this file and how the file is used.

5. SYSVAR 'S' controls the column layout for the body of the document. This macro defines &SYSVARS (style) to "one," "two" or "off" based on the value given on the command. (SYSVAR 'C' is treated as synonymous with SYSVAR 'S' for compatibility with the starter set in Release 1 of DCF.) The default is "one" for single column formatting.

6. SYSVAR 'T' controls whether or not the title page will be formatted and if so, whether it will be right aligned, left aligned or centered on the page. The macro defines &SYSVART (title page) to "right," "no," "left," or "center" based on the value given on the command. The default is "right."

7. SYSVAR 'X' controls whether or not the cross reference listing will be produced. The macro defines &SYSVARX (cross reference) to either "yes" or "no" based on the value given on the command. The default is "yes."

   DSM#SETV redefines a comment to reclaim its space in storage and to avoid reexecution on a second pass. This technique is explained in "Special Techniques" on page 13.

DSM#SETS

DSMPROF3 calls the DSM#SETS macro to initialize the &date and &time symbols and to define symbols for literal text strings. DSM#SETS performs the following processing:

1. Defines all of the literals (text strings) used in the starter set except for those in the DSM#MSG macro. All of the text strings are collected here to facilitate changing them.

2. Sets up the &date symbol to be of the form January 22nd, 1985 as follows:

   a. Puts the months of the year into the &* symbol array.

   b. Puts the endings for 1st, 2nd, and so on, into a local symbol, &*s.

   c. Calculates the position of the correct ending for the current day of the month.

   d. Selects the ending (st, nd, and so on) from the &*s symbol.

   e. Adds zero to the current day of the month to eliminate the possible leading zero[16].

   f. Adds zero to the current month to eliminate the possible leading zero[16].

   g. Builds the &date symbol using the month number as an index into the &* array that contains the months of the year. The statement that sets the &date symbol resolves as follows for the eighteenth day of the eleventh month:

```
.se date = '&*&*c &*b.&*a, 19&SYSYEAR
.se date = '&*&*c &*b.&*a, 1983
.se date = '&*&*c &*b.th, 1983
.se date = '&*&*c 18th, 1983
.se date = '&*2 18th, 1983
.se date = 'November 18th, 1983
```

[16] The day and the month will have a leading zero if they are less than 10. Performing arithmetic on the value removes the leading zero so that "03" becomes "3."

3. Sets up the &time symbol to be of the form 10:30 a.m. as follows:

   a. Determines if the hour is less than twelve and sets the &*m symbol to "a.m.." If the hour is twelve or greater than twelve &*m is set to "p.m.."

   b. Eliminates the possible leading zero on the hour for .a.m. by adding zero to it.[16]

   c. Subtracts 12 from the hour for p.m. to convert it from 24 hour time to 12 hour time.

   d. Sets the &time symbol.

4. This macro defines itself to "off" to reclaim its space in storage. This technique is explained in "Special Techniques" on page 13.

DSM#STYL

Column layout is defined by the DSM#STYL macro. This macro is also called by DSMPROF3 during initialization. It is called by DSMFRONT, DSMBODY, DSMAPPD, DSMBACKM and DSMINDEX to establish the column layout appropriate for each document section.

One parameter ("one," "two," or "off") can be passed to the macro. The parameter determines which of the 3 possible layouts will be set up. &SYSVARS, which defaults to "one," controls the layout of the body section. The DSM#STYL macro is also called by DSMBACKM and DSMINDEX for two-column layout regardless of the value of &SYSVARS. Similarly, the DSMFRONT macro requests one-column layout unless &SYSVARS is "offset," in which case offset layout is also used for the front matter.

If no parameters are passed, the value of &SYSVARS is used as the parameter. DSM#STYL does the following:

1. For one-column layout:

   a. Sets &@rc1 and &@rc2 to null. These symbols are used to establish the location of the revision codes around headings. See "Revision Codes for Headings" on page 76 for a detailed discussion of revision codes around headings. In the case of one-column format, no adjustment needs to be made so the symbols are set to null. The symbols are used in the APFs for heading levels two through four.

   b. Sets &@fn1 to 0. This symbol controls the left indention for footnotes. In the case of one-column layout no indention is desired.

      The &@fn2 symbol controls the right indention for the footnote. It is set to zero because there is no right indention for footnotes. This symbol is always zero in the starter set. It is provided exclusively to facilitate user modifications to footnotes.

   c. Defines a single column starting in position 0.

   d. Resets column line length to the current line length.

   e. Resets the .RC [Revision Code] ADJUST control word to the default.

   f. Heading level zero and one are aligned on the outside of the page if duplexing is active.

   g. Branches around the offset style definition to the footnote leader definition. See number 4. on page 34.

2. For two-column layout:

   a. Sets &@rc1 and &@rc2 to null. These symbols control the location of the revision codes around headings. For two-column layout the default location is used so these symbols are set to null. These symbols are used in the APFs for heading levels two through four.

b. Sets &@fn1 to 0. This symbol controls the left indention for footnotes. Footnotes are always formatted in one-column regardless of the column layout for the body of the page.

   The &@fn2 symbol controls the right indention for the footnote. It is set to zero because there is no right indention for footnotes. This symbol is always zero in the starter set. It is provided exclusively to facilitate user modifications to footnote.

c. Defines two-columns by:

   1) Calculating column line length by taking half of the line length minus the gutter space. The gutter amount is 4 spaces and is in the &@gutter symbol set in DSMPROF3.

   2) Calculating where the second column should start by adding the column line length to the gutter.

   3) Defining a two-column layout starting in position 0 and the position calculated above.

   4) Setting column line length. This will be 32 characters for line devices such as the 1403 and 3800 Printing Subsystem Model 1. For page printers, the column will be approximately 3.25 inches wide.

d. Resets the .RC [Revision Code] ADJUST control word to the default.

e. Heading level zero and one are aligned on the outside of the page if duplexing is active.

f. Branches around the offset style definition to the footnote leader definition. See number 4. on page 34.

3. For offset layout:

   a. Sets level zero and one headings to be left-aligned. This is necessary only if we have set up two- or one-column format when duplexing and then gone to offset style.

   b. Sets level two through four headings to cause section breaks.

   c. Sets vertical formatting to "top." This is done because we can't vertically justify offset style pages due to the numerous section breaks caused by headings.

   d. Performs calculations to determine the starting position for the text column and the position of the revision codes. The calculations are done in both spaces and device units because:

      - Rounding errors can occur when calculating spaces for the 3800 Printing Subsystem Model 1 in device units

      - We need to perform the calculations in device units for page devices.

      This is accomplished as follows:

      1) Assumes we are calculating in device units and sets up a local symbol for line length (&*ll) and for the revision code adjustment (&*two).

      2) Tests to see if we are formatting for a 3800 Printing Subsystem Model 1 and if so redefines the &*ll and &*two local symbols to be in spaces rather than device units.

      3) Calculates the width of one-fifth of the column—this will be the amount of offset.

      4) Subtracts the offset from the line length value (&*ll) to get the column line length value.

      5) Defines a local symbol, &*rc, to contain the revision code location. The revision codes are supposed to go all the way to the left of the page. Since they are placed relative to the beginning of the column, and we are going to offset the column, we

will need to move the revision codes to the left an amount equal to the offset plus 2.

6) Defines a local symbol, &*fn, to contain the starting position for footnote text. In offset style, footnotes line up with the text of the page.

7) Tests if we are formatting for a 3800 Printing Subsystem Model 1 and if not, resets the &*rc and &*fn local symbols to indicate that the values set are in device units.

e. Defines the &@rc1 symbol to have a value of ".rc adjust." This symbol is used in the APFs for level two through four headings to reset the location of the revision codes to the default value of two spaces to the left of the heading.

f. Defines the &@rc2 symbol to contain a value of ".rc adjust &*rc" which was calculated above. This symbol is used in the APFs for level two through four headings just after the heading control word is issued. It resets the revision code location relative to the left margin of the column rather than relative to the location of the heading.

g. Sets &@fn1 to line up with the text column. The &@fn1 symbol is used to create a left indent for footnotes.

   The &@fn2 symbol controls the right indention for the footnote. It is set to zero because there is no right indention for footnotes. This symbol is always zero in the starter set. It is provided exclusively to facilitate user modifications to footnote.

h. Defines one-column which starts one fifth of the way across the page. The space to the left (roughly 1.2 inches) is used to outjustify headings[17]. Column line length is set equal to the difference between the offset amount calculated above and the line length.

i. Resets the .RC [Revision Code] ADJUST control word amount equal to the space to the left of the text, roughly 1.2 inches [17].

4. For all layout formats, the footnote leader is defined using a horizontal rule (.HR [Horizontal Rule] control word). The rule name (@fnldr) and its length (&@fnldrlen) are both defined in DSMPROF3. The footnote leader is defined to start at the position saved in &@fn1. This lines the footnote up with the beginning of the text corresponding to the column layout that has been established.[18]

DSM#SET

DSMPROF3 calls the DSM#SET macro to initialize some very useful symbols. It performs the following functions:

1. Uses the &$PASS system symbol to set the &@lastpass symbol to "yes" or "no," respectively. This variable is used primarily in cross reference processing to tell if this is the last pass. The lines that define the &@lastpass symbol are somewhat complicated so a detailed explanation follows:

   a. The &@lastpass symbol is set to one minus its existence, times the value of the &$TWO system symbol, times three, plus one. This "magical incantation" results in a value of 4 for the first pass and a value of 1 for the second pass if the TWOPASS option was specified.

```
.se @lastpass = 1 - &E'&@lastpass * &$TWO * 3 + 1
```

[17] Since the calculations are done in device units (the smallest amount of space the output device is capable of moving) the exact amount of space will vary by device.

[18] One of the implications of this is that if you use a .SC [Single Column Mode] control word to switch from offset format to one-column format, the footnote leader and the footnotes will not move out to the left margin. To change column style and keep the footnotes lined up, this macro, DSM#STYL, should always be used.

When "TWO" is specified on the SCRIPT command, this line resolves as follows on the first pass

```
.se @lastpass = 1 - 0 * 1 * 3 + 1
.se @lastpass = 4
```

and as follows on the second pass

```
.se @lastpass = 1 - 1 * 1 * 3 + 1
.se @lastpass = 1
```

When TWO is not specified on the command, this line resolves as follows on the first (and only) pass:

```
.se @lastpass = 1 - 0 * 0 * 3 + 1
.se @lastpass = 1
```

Note: SCRIPT/VS evaluates arithmetic expressions from left to right without regard for operator precedence.

b. Finally, &@lastpass is set to either "yes" or "no" based on the calculation shown above. The "1" and "4" we calculated above refer to the position of the "yes" and "no" in the substr argument shown below.

```
.se @lastpass = substr 'yesno' &@lastpass 3
                        1  4
```

2. Defines the &rbl symbol to be the required blank (&$RB).

3. Initializes various counters and strings.

   a. The &@nest@l, &@nest@i and &@nest@q symbols are all set to 0. These are the nesting level counters for lists, imbeds and quotations, respectively.

   b. &@fig# and &@fn# are set to 1. These two symbols contain the number of the next figure and footnote, respectively. They are incremented in the DSMFCAP and DSMFTNT APFs.

   c. &@state is set to "open." This symbol is used to keep track of whether a footnote, list, quotation, title page, example, or figure has been started and not yet finished. During initialization it is set to "open" to indicate that nothing has been started yet.

   d. The &@sk@l symbol is set equal to &@sk@ls. This symbol controls the amount of skip before and after lists. The default value of &@sk@ls, set in DSMPROF3, is .75.

4. If cross referencing is in effect (&SYSVARX is "yes"):

   a. Defines a macro named IM to take over the function of the .IM [Imbed] control word. The IM macro calls the DSMIM macro, passing it the parameters on the control word line.

   b. Initializes the &@xref@d, &@xref@f, &@xref@h, &@xref@i and &@xref@n symbol arrays to a comment (.*). These symbol arrays are used to produce the cross reference listing. They are initialized to a comment just in case no IDs are used. If these symbols weren't set to a comment and there were no ids defined in the document, the symbols would be unresolved when they were printed by the cross reference listing macro. See "Cross-References" on page 147 for details on how these symbol arrays are used in cross referencing.

5. Initializes the symbol arrays used to write out the SYSVAR 'W' file (&@writ@d, &@writ@f, &@writ@h, &@writ@i, and &@writ@n) to a comment (.*) if SYSVAR 'W' has been specified. This is done just in case no IDs are saved for writing out. See "Cross-

References" on page 147 for details on how these symbol arrays are used in saving the cross reference information.

6. Defines &@it1, &@it2 and &@it3 to null strings if indexing has been requested. This is done to make sure that the symbols are not undefined when they are used on a .PI [Put Index] control word. The &$INDX system symbol is used to determine whether indexing has been requested or not.

7. Initializes the symbols used in the running heading and footing.

   a. The &@sec variable is initialized to null. This symbol may contain the security classification of the document. It is used in the running heading and on the title page.

   b. The &@stitle variable is initialized to null. This symbol is used in the running footing and contains either the title of the document or the most recent level zero or one heading.

   c. The &@shead variable is set to "&@stitle." The &@stitle symbol contains the most recent level zero or one heading or short heading. Its value is set up by the heading APFs.

8. Undefines itself to reclaim its space in storage. This technique is described in detail in "Special Techniques" on page 13.

Modifying Starter Set Initialization

Many modifications can be made to the initialization process that will effect the formatting of various elements throughout the document. A few of these are discussed below.

Creating Your Own Profile and Epifile

Instead of modifying the DSMPROF3 profile directly, you may want to create your own profile to use on the SCRIPT command. In order to take advantage of the initialization performed in DSMPROF3 you would want to start your profile by imbedding the starter set profile. Follow this with any of your own modifications.

Note: There is one exception: if you are going to override the line length (&$LL) used in the starter set, do so before imbedding DSMPROF3. The reason is that before DSMPROF3 changes the line length it checks to see if you have modified the line length and, if so, doesn't change it. The page layout defined in the DSM#STYL macro called by DSMPROF3 uses the line length. If you were to change the line length after processing DSMPROF3, you would need to call the DSM#STYL macro again to reset the page layout. By changing the line length, then imbedding DSMPROF3, you can avoid all this extra work.

To take advantage of the cross reference listing and imbed trace processing performed by the epifile in DSMPROF3, you would need to create an epifile in your own profile. Your epifile is invoked automatically by SCRIPT/VS at the end of processing. To process the epifile in DSMPROF3, imbed DSMPROF3 again. Your profile might look like this:

```
.ll 70
.im DSMPROF3
.
.
.
.ef
.im DSMPROF3
```

The second imbed of the profile causes the second portion of DSMPROF3 to be processed.

Changing the Format of the Date

The format of the &date symbol can be changed to the form "18 November 1983" instead of "November 18th, 1983" by rearranging the local symbols in the DSM#SETS macro.

```
    .se date = '&*b &*&*c 19&SYSYEAR
```

where

&*b	contains the day of the month
&*c	contains the number of the month
&*1 to &*12	contain the names of the months

The &*1 to &*12 symbols come into play when the &*c symbol is resolved to a number so that &*&*c becomes &*11 or whatever.

Since we are no longer using the &*a symbol which produced the date ending (th, nd, and so on) we can also delete the lines associated with defining &*a which includes the line setting up the &*s symbol.

Changing Default SYSVAR Values

The defaults for each of the SYSVARs are built into the processing in the DSM#SETV macro which is described in "Starter Set Initialization" on page 19. For a detailed explanation of how the SYSVARs are processed and how the defaults are set up, see "Special Techniques" on page 13.

For example, the default for SYSVAR 'D' is "no" duplexing. If you wish to change this to default to "yes," change the second line of the macro from

```
    .if &*a eq 0 .se *a = 1
```

to

```
    .if &*a eq 0 .se *a = 4
```

You can change the default for most of the SYSVARs in exactly the same manner.

Setting a Prefix for Level One Headings

To set up a specific prefix for level one headings in the body simply activate a line that is commented out in DSMPROF3:

```
    .*.se @bodyhead1 = 'Chapter
```

Set the symbol &@bodyhead1 to the text you wish to use as the prefix, such as "Chapter" or "Part" and remove the comment:

```
    .se @bodyhead1 = 'Chapter
```

The logic to perform the prefixing is already built into the DSMHEAD1 APF and the DSMBODY APF. It is necessary to use a :BODY tag to activate the prefixing because only level one headings in the body can be prefixed.

Alternatively you can define a symbol named LL@Chap in the DSM#SETS macro:

```
    .se LL@Chap 'Chapter
```

and then use this as the value of &@bodyhead1.

```
   .se @bodyhead1 '&LL@Chap
```

This is a better choice for setting &@bodyhead1 than hard-coding the word "Chapter."

Changing the SYSVAR 'W' File Name

The .DD [Define Data File-id] control words that define the name of the SYSVAR 'W' file are located in the DSM#WRIT macro. There are several different .DD [Define Data File-id] lines—one for each of the environments in which you can use SYSVAR 'W'. Change the .DD [Define Data File-id] control word to the new name for the environment you are interested in. For example, to change the filetype from DSMREFS to IDFILE in the CMS environment, change

```
    .if &$SYS eq CMS .dd dsmutwtf &SYSVARW DSMREFS
```

to

```
    .if &$SYS eq CMS .dd dsmutwtf &SYSVARW IDFILE
```

The same file names are built in the .DD [Define Data File-id] control words that define the SYSVAR 'R' file. These lines are located in the DSM#SETV macro where SYSVAR 'R' is processed.

```
   .if &E'&SYSVARR ne 0 .an &$SYS eq CMS
   .th .dd DSMUTREF &SYSVARR dsmrefs *
   .if &E'&SYSVARR ne 0 .an &$SYS eq TSO
   .th .dd DSMUTREF dsn &SYSVARR..DSMREFS
   .if &$SYS eq CMS .or &$SYS eq TSO .an &E'&SYSVARR ne 0 .im DSMUTREF
```

It is important that you change both the DSM#WRIT and the DSM#SETV macros.

Changing Spacing and Indention Settings

Many of the starter set tags involve spacing and indention functions. For example, a space is skipped before and after a long quotation and it is indented 3 spaces on both the left and the right. The amount of space skipped and the amount of indention are controlled by symbols set in DSMPROF3. By changing the value of the symbol you can change the amount of space or indention. The defaults are shown in Figure 5 on page 21. The symbols are set at the beginning of the profile using .GS [GML Services] ARGS and .GS [GML Services] VARS control words.

```
   .gs args 10    2     4     4     0     3     4     4     2     0
   .gs vars @in@d @in@f @in@z @in@o @in@p @in@q @in@s @in@u @in@x @in@g
```

To change the amount of indention for a long quotation to 4 characters all you would need to do is change the "3" in the first line shown above, to a "4."

Changing the Rules Used for Figures

By default, figures are framed with rules at the top or the bottom or both, depending on where the figure is placed. The rules are normally drawn in highlight font 2 or, for page printers, in the default rule. If a box is requested, it is also drawn in highlight font 2 or, for page printers, the default rule.

If you are using a monospaced body font and a proportional bold font for the 3800, this can create problems. You will need to change the font used to draw rules and boxes because SCRIPT/VS cannot use both kinds of fonts simultaneously. To do this you need to activate the .DR [Define Rule] control word that is in the profile.

```
.dr @figrule
```

The @figrule rule name is specified on both the .HR [Horizontal Rule] control word line and .BX [Box] control word line in the APF for :FIG. However, because of the way in which the @figrule rule is defined (that is, with no parameters specified) it will use the default rule for page printers and will use the current font for the 3800 Printing Subsystem Model 1. We change to highlight font 2 before printing the box which produces bold rules on the 3800 Printing Subsystem Model 1.

By specifying a definition on the .DR line in the profile you can change the rule that is drawn. For example, to draw the boxes and rules in the body font for the 3800 change it to

```
.dr @figrule font &$CHAR(1)
```

To change it to a slightly thicker than normal rule for page printers, change the rule definition to:

```
.dr @figrule weight .4mm
```

Both the font and weight parameters can be specified simultaneously on the .DR [Define Rule] control word. If both are specified, SCRIPT/VS will use the one that applies to the device that we are formatting for.

```
.dr @figrule weight .4mm font &$CHAR(1)
```

If you specify both WEIGHT and FONT on the .DR [Define Rule] control word, and you are using &$CHAR(2) as the font value, be careful to put the WEIGHT parameter first. This is necessary because when you format for a page printer the value of &$CHAR(2) may well be null which will cause an error on the .DR [Define Rule] control word line unless it is at the end of the line.

Changing Fonts for Figures

The starter set figures do not perform a font change for the body of the figure. For page printer output, you may wish to have figure text set in a monospaced font (as we have done in this book), depending on the content of the figure. To accomplish this all you have to do is activate the font definition for the "figfont" font in DSMPROF3.

For example, change

```
.*df figfont
```

to

```
.df figfont type('prestige elite')
```

The initial definition is commented out, which will cause figure text to be set in the default font. You may specify any font on the .DF [Define Font] control word. The font is started by the DSMFIG macro and the previous font is restored by either the DSMFCAP, DSMFDESC or DSMEFIG macro, whichever is processed first.

Note: There are two different "figfont" font definitions—one for the 4250 printer and one for the IBM 3820 Page Printer and 3800 Printing Subsystem Model 3. You may want to change both of these or just one.

Changing Font Definitions

All of the fonts for the starter set are defined in the profile. You can change the definitions by changing the .DF [Define Font] control word line. You might want to refer to the description of the .DF control word in *Document Composition Facility: SCRIPT/VS Language Reference* before doing this. Notice, however, that the fonts are defined differently for each logical device. For some devices the font definitions are further refined to be based on the number of fonts that were specified on CHARS as well as the device.

For example, suppose you wanted to use a bold body size font for the running headings and footings when formatting for a page printer. The default is to use a 9-point bold italic font for the heading and a 9-point bold font for the footing:

```
.df @rh type(bold italic 9) up
.df @rf type(bold 9)
```

You could simply change these lines in DSMPROF3 to:

```
.df @rh type(bold)
.df @rf type(bold)
```

The profile, however, does not define what typefaces to use for the 4250 printer or for the 3800 Printing Subsystem Model 3. The typeface is set by the default font or by what is specified with the CHARS option on the SCRIPT command. The default typeface is Monotype Times New Roman[19] for the 4250 printer, unless it has been changed by your installation. For the IBM 3820 Page Printer and 3800 Printing Subsystem Model 3, Sonoran Serif[20] is the default typeface.

You may override the default font by simply specifying a different font on the SCRIPT command with the CHARS option. However, the starter set font definitions for page printers are set up based on an assumption that the initial font is a typographical font which comes in many sizes and styles.[21] If you specify a typewriter font or some other font which does not come in the full set of sizes and styles, you may get some SCRIPT/VS error messages regarding fonts.

Creating a New Highlight Level

The starter set provides three levels of highlighting (:HP1 through :HP3). This may not be enough, especially if you are using a page printer. To create a fourth level of highlighting you will need to do several things to create a new tag named :HP4.

1. Map the :HP4 tag to an appropriate APF.

   ```
   .aa hp4 dsmhp4 (noatt) dsmehp
   ```

   We can use the same APF for the end tag as all the rest of the highlight tags —DSMEHP because all this APF does is a .PF [Previous Font] control word and that's all we will need to do here.

[19] Trademarks of The Monotype Corporation, Limited.

[20] Data derived under license from The Monotype Corporation, Limited.

[21] For the 4250 Printer, the Excelsior typographical font does not contain any bold italic versions, although it does come in a full range of pointsizes. This means that it can not be used as the default font for the starter set unless some modifications are made to the font definitions in the profile.

2. Define a hi4 font for all devices that it applies to.

   ```
   ...38PP
   .df hi4 type(apl 10) codepage t1s0ae10
    .
    .
    .
   ...4250
   .df hi4 type('light italic' italic) codepage aftc0293
   ```

   These lines define hi4 to be the APL font (named "light italic" for the 4250 printer and "APL" for the IBM 3820 Page Printer and 3800 Printing Subsystem Model 3). The pointsize for the highlight font will come from the current font. The APL font only comes in a few selected sizes which could result in an error message if it is used somewhere other than in body text. The codepage names (AFTC0293 and T1S0AE10) also must be specified because the APL fonts use a different codepage arrangement than the normal text fonts. These codepages will not work with the text fonts which means that it is important to either redefine all of the text fonts to specifically use the correct text codepage or to always start and end the APL highlight font without nesting any other fonts inside of it.

3. Write a new APF named DSMHP4 to process the :HP4 tag.

   ```
   .bf hi4 =
   ```

   By putting the equal sign on the end of the .BF [Begin Font] control word line you instruct SCRIPT/VS to restart the current font whenever the hi4 font isn't defined or can't be started. This allows you to format to all devices without getting an error message and a real font change will occur only for the page printers.

Modifying the Running Heading or Footing

One of the most common modifications that is made to the starter set involves changing the running heading or footing. The definitions for the heading and footing are in DSMPROF3. One way to override these definitions is to create you own profile that imbeds DSMPROF3 and then redefines the running heading and footing in your profile. You could also modify the profile (or a copy of it) directly.[22]

Within the running heading and footing definitions you may perform almost any processing you wish. Consult the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for more information about headings and footings.

For the starter set the heading and footing definitions are the same for the entire document. Depending on your application you may find it necessary to vary your running heading or footing according to what section of the body you are in. You will have to redefine the heading and footing to be a function of the document section. You will have to add the definitions to the DSMFRONT, DSMBODY, DSMAPPD, DSMBACKM and DSMINDEX macros. This way each of the definitions can be tailored to the document section. The basic running heading and footing definition should remain in the profile and should reflect the style chosen for the body of the document. This is recommended because when documents are created without any front matter users rarely use the :BODY tag. In these cases you would want the profile to define the body style for running headings and footings.

---

[22] The GML starter set is a fully supported part of the Document Composition Facility program product provided that neither the profile nor the macro library have been modified in any way. What this means is that you should be careful to not alter the base version of these files but rather should make your own copies or user libraries.

Changing the Page Dimensions

SCRIPT/VS has built into it a set of page dimensions that are a function of the logical device you are formatting for. The starter set, with the exception of the line length and column line length parameters, does not alter these dimensions in any way. The page margins will be a function of either the default bind for the device or the value of the BIND option on the SCRIPT command. The page width will be the default page width for the logicial device, as will be the top and bottom margins for the page. You can find out more about the page dimension specifications in the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide.*

Line length and column line length in the starter set are modified depending on the column layout (style) of the document. For one-column style, the default line length is used and column line length defaults to line length. For two-column and offset style the line length is lengthened to 6.8 inches in DSMPROF3. Column line length is then set in the DSM#STYL macro to provide two short columns or an offset one.

To change line length you can either modify the profile or create your own profile that modifies line length before imbedding DSMPROF3. See "Creating Your Own Profile and Epifile" on page 36 for more details on doing this.

Modifying column line length for two-column or offset style involves modifying the .CL [Column Line Length] control word line in the DSM#STYL macro. For two-column style you can also control the width of the gutter between the column by changing the value of the &@gutter symbol that is defined in DSMPROF3.

Creating Three Column Format

The starter set provides easy access to one-column, two-column and offset style column layouts. If we want to create a three column layout we will need to make some decisions about how it should look and then modify the DSM#STYL macro. For example, suppose that we wanted to create columns that were 2 inches wide with a gutter of .4 inches between each of them. This happens to coincide with a line length of 6.8 inches which is also the line length the starter set uses for two-column formats. Suppose additionally that we wanted to be able to specify on SYSVAR 'S' that we wanted three columns. This means that we will also need to modify the DSM#SETV macro which is the macro that processes the SYSVARs. We will have to modify it to recognize a value of "three."

Let's do this first, since it's easier. SYSVAR 'S' is currently processed by the following lines in DSM#SETV:

```
.se *a       = index '-1---2---OFFSET-ONE-TWO-' '-&U'&SYSVARS.'
.if &*a eq 0 .se *a = 1
.se SYSVARS = substr 'one two off    one two' &*a 3
```

These lines are explained in "Special Techniques" on page 13. The first line determines what the value of SYSVAR 'S' is. The second line sets up a default value of "1" if SYSVAR 'S' either was specified or was invalid. The third line resets SYSVAR 'S' to a standard lower case value that will be easy to test elsewhere in the starter set macros.

We will need to include "THREE" and "3" in line 1 as valid values.

```
.se *a       = index '-1---2---3---OFFSET-ONE-TWO-THREE' '-&U'&SYSVARS.'
```

Next we will have to include "thr" in the third line as a standard value for SYSVAR 'S'. We had to use "thr" instead of "three" because we were using only the first three letters of each style as the new value.

```
.se SYSVARS = substr 'one two thr off    one two thr' &*a 3
```

Notice that we had to add the "thr" to the third line in the same relative positions that we put "3" and "THREE" in the first line.

The next part of the modification involves modifying the DSM#STYL macro to include a section for defining the three column layout. The lines to do this will be very similar to the lines used to set up two-column format. The label that precedes the section will be "thr" because we will get there by branching to a label whose name is in &SYSVARS.

```
.if &L'&*1 eq 0 .go &SYSVARS
.el .go &*1
```

The control word lines in the "thr" section should go after the end of the processing for two column format and should look like this:

```
...two
 .
 .
.go fnldr
...thr
.gs args ''   ''   0   0
.gs vars @rc1 @rc2 @fn1 @fn2
.cd 3 0 2.4i 4.8i
.cl 2i
.if &SYSVARD eq yes .dh 0 outside
.th .dh 1 outside
.go fnldr
```

This is exactly the same as the lines used to set up two-column except that we've dropped the calculation for the column line length and the column position. We've hard coded these instead.

Other Modifications to the Profile

You can modify just about anything in the profile. That's why the profile is there. Some of the other modifications that you can make include:

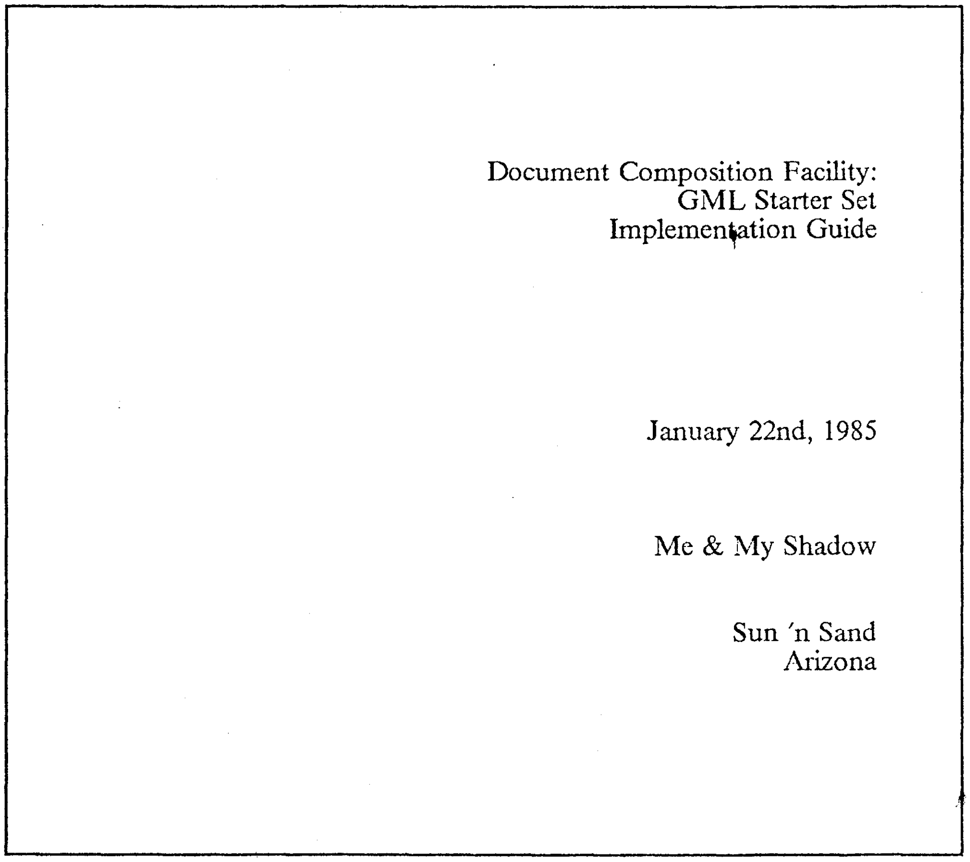
- Changing the heading definitions which is discussed in "Modifications to Headings" on page 82
- Changing the default figure placement and width which are described in "Modifications to Figures and Examples" on page 122
- Changing the definition of the footnote leader which is discussed in "Modifications to Quotes, Notes, Footnotes and Highlights" on page 134
- Changing list item identifier definitions which is discussed in "Modifications to List Processing" on page 108.

Title Page

Overview

There is a set of macros that handles the tags to define and create the title page. Only the :TITLEP tag and the :ADDRESS tag[23] are associated with their respective APFs in DSMPROF3. The APFs for the :TITLEP tag enables the rest of the tags used to create the title page. The actual title page is formatted by DSM#TIPG which is called from the DSMETTLP APF.

Here's how a typical title page would look:



Document Composition Facility:
GML Starter Set
Implementation Guide

January 22nd, 1985

Me & My Shadow

Sun 'n Sand
Arizona

Figure 9. Sample Title Page: This is a sample of what the default title page for the starter set looks like. The default formatting is to align the text on the the right hand side of the page.

[23] The :ADDRESS tag is enabled in DSMPROF3 because it can appear other than on the title page.

The APFs described here are for the following nine tags, which together describe a title page:

```
:TITLEP
    :TITLE
    :AUTHOR
    :ADDRESS
       :ALINE
    :EADDRESS
    :DATE
    :DOCNUM
:ETITLEP
```

With the exception of the APFs for the :TITLEP and :ETITLEP tags all of the APFs for these tags save the residual text for processing later. This is why the title page appears the same, regardless of the order in which the individual tags are entered. This is also what makes it so easy to add pieces of information to the title page. See "Modifications to the Title Page" on page 52 for details about how to do this.

Initialization

During initialization, which is described in detail in "Starter Set Initialization" on page 19, several things are done to allow the title page to be formatted correctly.

DSMPROF3

DSMPROF3 maps the :TITLEP and :ETITLEP tags to the DSMTTLEP and DSMETTLP APFs, respectively. The :ADDRESS and :EADDRESS tags are also mapped to the DSMADDR and DSMEADDR APFs, respectively. All other tags are enabled later.

The running heading is defined in DSMPROF3 and uses the &@sec symbol. The &@sec symbol contains the security classification as entered on the SEC attribute of the :GDOC tag. This symbol is also used on the title page.

DSM#SETS

The DSM#SET macro defines symbols including &date and &LL@DocNm ("Document Number") that can be used on the title page.

DSM#SETV

The DSM#SETV macro processes the system variables (SYSVARs) given on the SCRIPT command. Two of these system variables are relevant to the title page:

1. SYSVAR 'D': Indicates whether or not the document is to be duplexed. It is not directly used by the title page macros, but the setting of SYSVAR 'D' affects how the value of the STITLE attribute of the :TITLE tag is used.

   If duplexing is in effect, the short title given on the STITLE attribute is used in the running footing on even pages. If we're not duplexing, the short title appears in the running footing only when there has been no level zero or one heading entered.

2. SYSVAR 'T': Indicates whether or not the title page is to be printed, and, if so, whether or not it should be left, right, or center aligned on the page. The value of the &SYSVART symbol is set to "right," "center," "left," or "no."

If SYSVAR 'T' is not specified on the command, the default is to print a title page right justified as in Figure 9 on page 45. See "Modifying Starter Set Initialization" on page 36 for details about how to modify this default setting.

Title Page Tag Processing

DSMTTLEP

The DSMTTLEP APF processes the :TITLEP tag which is always the first tag for the title page. It establishes the environment for the title page structure as follows:

1. The &@state symbol is set to "TtlPg" to indicate that a title page is being defined. This symbol is used repeatedly to determine if we are still within the title page structure defined by :TITLEP and :ETITLEP.
2. The title page tags are mapped to the appropriate APFs (:AUTHOR to DSMAUTHR, :DATE to DSMDATE, :DOCNUM to DSMDCNUM and :TITLE to DSMTITLE). These tags are valid only within the title page structure.
3. Various title page symbols and symbol arrays are initialized:

   a. &@addctr array for counting addresses

   b. &@author array for saving the authors' names

   c. &@address array for saving the names of the address arrays

   d. &@docnum for saving the document number

   e. &@docdate for saving the document date

   f. &@title array for saving the document title lines.

   These are initialized to null values in case the user does not specify them. If this is not done, the symbols will appear unresolved on the title page rather than as null (nothing).

DSMTITLE

The DSMTITLE APF processes the :TITLE tag. Multiple :TITLE tags can be used to enter multiple lines of the title, but only one should have an STITLE attribute on it. The DSMTITLE APF performs the following processing:

1. Saves the residual text of the tag as the next element of the &@title array, which will be printed on the title page.
2. Changes the symbol array separator to a blank because we may put the entire contents of the &@title array into the &@stitle symbol for use in the running footing.
3. Determines whether or not &@stinit exists. &@stinit will exist only if an STITLE attribute was specified on a previous :TITLE tag. If &@stinit does not exist, no short title was found on a previous :TITLE tag.

   If there is no short title, we'll need to use the full title. So we'll set &@stitle to the entire &@title array using a blank as the array separator. If this tag or a subsequent tag has an STITLE attribute, &@stitle will end up getting reset to the short title.
4. Calls the DSM@STTL macro to process the STITLE attribute, if it is present. This resets &@stitle to the attribute's value.

DSM@STTL

The DSM@STTL macro processes the STITLE attribute of the :TITLE tag. It is called by the DSMTITLE APF only if the STITLE attribute is present. The value of the attribute is used in the running footing for even pages, if duplexing, and for all pages if no level zero or one headings are entered. DSM@STTL does the following:

1. Sets &@stinit to 1 to indicate that a short title attribute has been entered. This symbol is used by the DSMTITLE APF to determine if an STITLE attribute was specified.

   A separate variable (&@stinit) is required to indicate the existence of a short title because the &@stitle variable must always exist. It was set to a null value in the DSM#SET macro during initialization to prevent errors in the running footing if there were no STITLE attributes or :TITLE tags. The DSMTITLE APF also sets &@stitle in case there is no STITLE attribute.

2. Saves the attribute value in the &@stitle symbol.

   It is possible to have more than one :TITLE tag. Hopefully, only one of them will have an STITLE attribute. If there is more than one STITLE, they will all get processed, but the value of the last STITLE is used.

DSMDATE

The DSMDATE APF processes the :DATE tag. If there is residual text on the tag, it means that the user has supplied his own date rather than used the processing date. In this case the residual text is saved in the &@docdate symbol and is also put into the &date symbol.

If there is no residual text, the current date (&date) is put into the &@docdate symbol, which will be printed on the title page. The current date was defined in DSM#SETS during initialization.

Either way, &@docdate and &date end up the same if the :DATE tag is used.

DSMAUTHR

The DSMAUTHR APF processes the :AUTHOR tag. It saves residual text as the next element of the &@author array, which will be printed on the title page. Since an array rather than a simple symbol is used, more than one :AUTHOR tag may be use.

DSMADDR

The DSMADDR APF processes the :ADDRESS tag. Because this tag can appear on the title page or in text, two different processes are required. DSMADDR does the following:

1. Maps the :ALINE tag to the DSMALINE APF. This tag is not really valid or necessary when an address is being formatted inline with text rather than on the title page. The mapping is done anyway to prevent the user from getting an error message if he forgets this and uses the :ALINE tag for an inline address.

2. Uses the .GS [GML Services] SCAN control word to obtain the residual text. If there is residual text, it is considered to be the first line of the address.

3. Checks the &@state symbol to determine if a title page is currently being defined.

   If we are on the title page, an array is set up for each address. The name of the array is &@aline&@addctr where &@addctr is incremented each time we encounter an :ADDRESS tag.

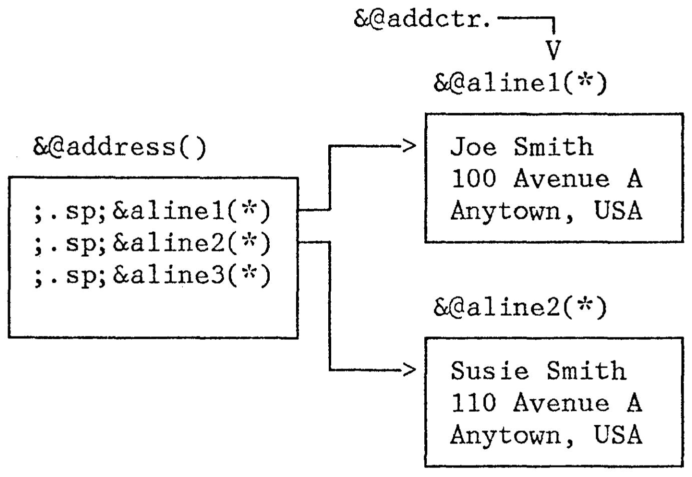


Figure 10. The Format of the Address Arrays

a. &@addctr is incremented for each address, so the first address will be put in an array named &@aline1(), the second will go into &@aline2(), and so on. See Figure 10 on page 49.

b. The array is undefined first (set "off") to get rid of any previous address.

c. The name of the array, preceded by a .SP [Space] control word, is saved in the &@address() array that will be printed by DSM#TIPG when the :ETITLEP tag is encountered. The .SP [Space] control word generates a blank line between each address on the title page.

d. Any residual text from the :ADDRESS tag is put into the first element of the &@aline&addctr array.

4. Formats the address as a simple compact list, if the :ADDRESS tag is found outside of the title page. This is done as follows:

   a. Skips &@sk@s (set by DSMPROF3 to .75).

   b. Saves the current environment because we're going to change the formatting mode and we don't want to have to specifically restore it.

   c. Turns formatting off with .FO [Format Mode] OFF. This means that each input line that follows, before the :EADDRESS tag, will become an output line. This is what makes the :ALINE tag unnecessary for inline addresses.

   d. Indents +&@in@s, which is set in DSMPROF3 to 4. We are using an incremental indention value rather than an absolute value because we have no idea whether there is any current indention or not.

   e. Begins a keep if &@state is "open," indicating the middle of open text rather than in a figure, a list, a footnote, and so on. If this is not open text, a keep should not be started because there is probably a keep or float already in progress. Starting a keep within a keep causes the user to get an error message.

f. Exits the macro here, if there was no residual text. Otherwise, the residual text is formatted in literal mode. Literal mode is used so that initial periods and semi-colons are treated as punctuation marks, rather than special delimiters for control words.

DSMALINE

The DSMALINE APF processes the :ALINE tag as follows:

1. This tag is not necessary outside of the title page. If we are not currently defining a title page (&@state is not "TtlPg"), the macro ends because there is nothing to do. The residual text is formatted in format "off" mode by SCRIPT/VS as a simple list. The environmental changes to do this are set up by the DSMADDR APF. See number 4. under DSMADDR for details.
2. If we are on the title page (&@state is "TtlPg"), the residual text is saved in the next element of the current address array (&@aline&@addctr).

DSMEADDR

The DSMEADDR APF processes the :EADDRESS tag. It ends the address section as follows:

1. Remaps the :ALINE tag to the DSM#CNTX APF because :ALINE is not valid outside of an address structure.
2. Ends the macro here if the address is being defined for the title page (&@state is "TtlPg").
3. Ends the keep if we're not on the title page and the address is being formatted in open text (not in a figure, footnote, or example).
4. Restores the formatting environment.
5. Performs a conditional skip.

DSMDCNUM

The :DOCNUM tag is processed by the DSMDCNUM APF which saves the residual text in &@docnum, which will be printed on the title page.

DSM@SEC

This macro is not really part of the title page macros, but is included here because it saves the security classification attribute value in the &@sec symbol that is used on the title page. It is invoked by the DSMGDOC APF to process the SEC attribute of the :GDOC tag if it is present.

The security classification (&@sec), if entered, appears highlighted on the title page. It is also put into the running heading.

DSMETTLP

The DSMETTLP APF processes the :ETITLEP tag. It ends the title page definition as follows:

1. Checks to see if a title page is currently being defined as indicated by &@state. If we're not in a title page structure, a message is issued that a title page end tag was found outside of the title page and the macro ends.
2. Sets &@state to "open" to indicate that there is no current special structure being defined.

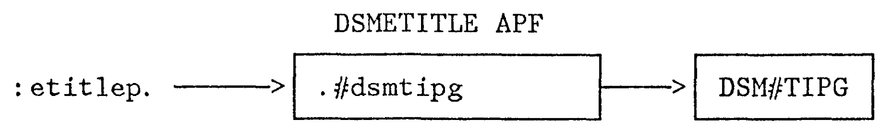


Figure 11. Producing the Title Page

3. Calls the DSM#TIPG macro to format the title page unless the value of &SYSVART is "no" indicating that no title page is wanted.

4. Remaps the :AUTHOR, :DATE, :DOCNUM and :TITLE tags to the DSM#CNTX APF.

Producing the Title Page

DSM#TIPG

The title page is formatted using the information saved by the title page APFs described above. Because all of the variables we're going to use on the title page were initialized to null by the DSMTTLEP macro, we don't have to worry about whether or not the user specified them. If they weren't specified, they will appear as null (or nothing).

Each of the pieces of information that goes on the title page has a special font defined for it for page printers. The font definitions are all in DSMPROF3. In most cases, there is no comparable font change that we can do for a line printer, so we keep restarting the current font. DSM#TIPG performs the following processing:

1. Suppresses running headings and footings as these would be undesirable on a title page.

2. Performs a conditional page eject with a .CP [Conditional Page Eject] control word because we want an entire new page for the title page.

3. Saves the current formatting environment.

4. Resets the line spacing ranges (.LS [Line Spacing] control word) for expansion and compression for vertical justification to 1.0. In other words, we have turned off vertical justification by eliminating the ranges on the .LS [Line Spacing] control word. The original ranges were established in the profile.

5. Establishes a single column page layout. We did not use the DSM#STYL macro because it would have done more than we wanted. All we needed to do was use the .SC [Single Column Mode] control word to get the desired results.

6. Turns off spelling checking because there might be some unusual things on the title page. The text will have already been verified when the tags were first encountered.

7. Turns off hyphenation because we wouldn't want anything to be hyphenated on the title page.

8. Restores the control word separator to the default setting (;) with the .DC [Define Character] CW control word. We constructed the &@address array using a semicolon as a control word separator and we need to make sure that's what it is when we process the array. See Figure 10 on page 49 for a example of what the &@address array looks like. We won't need to worry about setting it back to whatever it was before this because we are going to restore the entire environment when we're all done.

9. Uses the value of &SYSVART, which can be "right," "left," "center," or "no" as the .FO [Format Mode] parameter to set up the formatting mode. .FO [Format Mode] NO wouldn't work but we don't need to worry about this combination because we won't get this far if "no" was specified.

10. Leaves two inches of space at the top of the page before the title.

11. Changes the symbol array separator to cause a break between the elements in the numerous title page symbols. This is done in two steps because the array separator is limited to four characters:

    a. The &@ symbol is set up to be a .BR [Break] control word.

    b. The characters used to construct the symbol are used as the array separator. This is equivalent to setting the array separator to ";.br;", but we cannot set it directly because we can only use four characters.

12. Changes to the title font or the highlight level 2 font, if the title font is undefined.

13. Increases line spacing to some factor of normal. The factor is in &@ttllo which is set in DSMPROF3 to be 1.2. The reason we increase the line spacing here is that on page printers when a large font is used, as is the default on the title page, the line spacing appears to be too small. By increasing the line spacing by 20%, greater line separation is achieved. The factor of 1.2 rounds down to 1.0 on line devices, so there is no conflict between devices here.

14. Formats the entire title array. Each element of the &@title array contains one line of the title of the document. All we want between them is a line break. This is why we defined the array separator to be a .BR [Break] control word.

15. Resets line spacing and restores the previous font. Two inches of space is skipped.

16. Formats the following symbols and arrays with extra space between each one. Each one has a special font for use on page printers and the current font is used for other devices. The symbols and arrays are:

    a. Document number (&@docnum) preceded by the words "Document Number" (&LL@DocNm)

    b. Document date (&@docdate)

    c. Author array (&@author(*))

    d. Address array (&@address(*)). Each element in the &@address array is the name of an array that contains a set of address lines. See Figure 10 on page 49 to see exactly what it will look like.

    e. Security classification (&@sec). This is printed in literal mode for one line to protect against semicolons getting treated as control word separators, and so on.

17. Restores the previous formatting environment along with the running heading and footing. We have saved and restored the environment because we've changed things like the formatting mode, the array separator, and the control word separator, and we do not want to have specifically reset them all. It is easier to restore the whole environment.

18. Undefines the macro. This has the effect of reclaiming the storage that has been used to hold the macro lines. We do this because the DSM#TIPG macro is only used once. See "Special Techniques" on page 13 for a full explanation of this technique.

Modifications to the Title Page

Changing Default Title Page Formatting

Like everything else in the starter set, there are many things that can be modified for the title page. You can modify the default value for &SYSVART from right justified to centered. This requires a simple modification to DSM#SETV and is described in "Modifying Starter Set Initialization" on page 36.

Changing Spacing

Another possible modification is to change the spacing between the various elements on the title page. This requires changing the space unit values on the .SP [Space] and .SK [Skip] control word lines in the DSM#TIPG macro.

Adding a Box

To put a box around the title page, we need to add a .BX [Box] control word at the beginning of the DSM#TIPG macro to start the box. We can create the box in the first and last positions of the line and indent the text for the title page to make sure that it will not overlap the box.

We don't know if the text will be formatted on the right or left. We can either indent both sides with .IN [Indent] and .IR [Indent Right] control words or we can test &SYSVART to figure out which side it will be on and then adjust only that side. It's much easier and quicker to adjust both sides.

Here are the lines we would need to start the box:

```
.bx left right
.in 2
.ir 2
```

These three lines should be added to the DSM#TIPG macro right after the .CP [Conditional Page Eject] control word at the beginning of the macro.

Then we need to close the box at the end of the DSM#TIPG macro—just after .SK 3, following the security classification. We can use the &$LC system symbol to calculate how much space is left before the bottom of the page. The &$LC symbol gives us an *approximation* of the number of lines left on the page. Because we'll want to put one more line on the page (the end of the box), we'll want to space down to one less than the number of lines left.

However, &$LC isn't always totally reliable because sometimes lines are partially processed and we think they are already on the page, but they haven't been counted yet and won't be reflected in &$LC. In this case, if we do a break we can use &$LC fairly reliably. The break causes any lines that are partially processed to be completed.

Here are the lines we would need to close the box.

```
.br
.se *a =  &$LC - 1
.sp &*a
.bx off
```

We have calculated how much to space (&*a), spaced that amount and then ended the box.

Adding Existing Information

Adding information that already exists to the title page is very easy because of the way the title page is formatted. Suppose that we wanted to add the time to the page. This one is particularly easy because the information we want is already available in a symbol (&time). All we would need to do is decide where we wanted it and add it to the DSM#TIPG macro along with another .SP [Space] control word to leave space around it.

You might also want to select a font for the time information if you were formatting for a page printer. In this case we would need to also add .BF [Begin Font] and .PF [Previous Font] control words to the DSM#TIPG macro and define the font in DSMPROF3.

Adding New Information

Suppose we wanted to add an entirely new piece of information, such as a reviewer's name prefixed with the word "REVIEWER:." We would need to create a whole new tag. This involves the following steps:

1. Pick a tag name (:REVIEWER).

2. Add it to DSMPROF3 mapped to the DSM#CNTX APF because we don't want it used except on the title page.

   ```
    .aa reviewer dsm#cntx (noatt)
   ```

3. Create a symbol called &LL@revwr in the DSM#SETS macro and set it to "REVIEWER:."

   ```
   .se LL@revwr = 'REVIEWER:
   ```

   We'll use this literal to prefix the text of the tag. We could just as easily use the actual word instead of a symbol, but because all the other literal strings are kept together in one macro, we should continue this practice.

4. Enable the :REVIEWER tag in the DSMTTLEP APF by mapping it to a APF named REVIEWER.

   ```
   .aa reviewer reviewer (noatt)
   ```

5. Disable the :REVIEWER tag in the DSMETTLP APF by mapping it back to the DSM#CNTX APF.

   ```
    .aa reviewer dsm#cntx
   ```

6. Write an APF named REVIEWER and add it to the maclib. (See "Appendix A. Modifying the Macros" on page 171 for more details on how to do this.) The APF only needs to be one line long:

   ```
   .gs scan @reviewr
   ```

   This will save the residual text in a symbol named &@reviewr.

7. Add the lines necessary to format the reviewer information to the DSM#TIPG macro. This involves testing if the &@reviewr symbol exists. If it does not, the symbol was not set and we don't want to print it. If the symbol has been set, we want to print it with the word "REVIEWER:" before it.

   ```
   .if &E'&@reviewr eq 1 &LL@revwr &@reviewr
   .sp 2
   ```

This allows for a single reviewer's name on the title page. What would we need to do to handle more than one reviewer? First of all we would have to create another literal constant which would be "REVIEWERS:" in DSM#SETS

```
.se LL@revws = 'REVIEWERS:
```

so we could label them properly. We would still want the other symbol ("REVIEWER:") in case we only had one name.

Then we would need the REVIEWER APF to scan for the residual text and put it into a local symbol. Then this symbol is used to set up the next element of the &@reviewr array.

```
.gs scan *name
.'se @reviewr() = '&*name
```

The last thing we need to change is the DSM#TIPG macro. It would need to test element zero of the &@reviewr array to see how many names we have. If it is one, we would use the literal constant that represents REVIEWER:. If it is greater than one, we'd need the plural constant, REVIEWERS:.

```
.if &@reviewr(0) eq 1 &LL@revwr. &@reviewr(1)
.if &@reviewr(0) gt 1 &LL@revws. &@reviewr(*)
```

It might also be necessary to consider the amount of space that would normally be used here. It might be a good idea to reduce the blank space in other parts of the page to prevent the text of the title page from routinely exceeding a page.

Printing Two Dates

Another modification that you might want to do is to print both the current date and any document date the user might have specified.

We want the current date to be preceded with "Formatted on" or "Printed on" and we want both dates available to users in a symbol. We are going to have to do the following:

1. Create a symbol in DSM#SETS with the text we want.

   ```
   .se LL@Prnt 'Printed on
   ```

2. Modify the date APF (DSMDATE) to keep the document date in the &@docdate symbol and the formatting date in &date.

   The APF currently looks like this:

   ```
   .gs scan @docdate
   .'if &L'&@docdate eq 0 .'se @docdate '&date
   .'el .'se date '&@docdate
   ```

   We need to change it to:

   ```
   .gs scan @docdate
   .'if &L'&@docdate eq 0 .'se @docdate '&date
   ```

   This way we put the formatting date or the date the user gave us into the &@docdate symbol and still maintain both dates when they are present.

3. Alter the DSM#TIPG macro to print the &date symbol as well as the &@docdate symbol. This is the last step. The basic approach to how to do this is described in "Adding Existing Information" on page 53. However, we will have to include the &LL@Prnt symbol with the &date symbol to indicate that the date shown is the date the document was printed on.

Document Sections

Overview

A general document is composed of several different sections each of which contains different types of material. Each section is also formatted somewhat differently.

The following tags identify these sections:

```
:GDOC
   :FRONTM
      :TITLEP
      :ABSTRACT
      :PREFACE
      :TOC
      :FIGLIST
   :BODY
   :APPENDIX
   :BACKM
      :INDEX
:EGDOC
```

The primary function of the APFs for these tags is to separate the sections of the document with a page eject and perform any special processing necessary to begin that part of the document. Sometimes this includes changing the page layout or redefining the way headings are handled. These APFs also control the style of page numbering and the text of the running footing. Sometimes headings are generated for the sections. The section tags have no specific text associated with them and, except for the :GDOC tag, there are no attributes.

The title page section is slightly different from the others and involves many tags and macros. It is discussed fully in "Title Page" on page 45.

The index section, which is produced by the :INDEX tag, is described in "Indexing" on page 137.

The remaining document section tags are described below.

Document Section Macros

DSMGDOC

The DSMGDOC APF processes the :GDOC tag. The SEC attribute is processed by the DSM@SEC macro. The DSMGDOC APF then reclaims its own storage by undefining itself because this macro is used only once. See "Special Techniques" on page 13 for details about how storage is reclaimed.

DSM@SEC

The SEC attribute of the :GDOC tag is processed by the DSM@SEC macro. The value of the attribute is saved in the &@sec symbol which is used on the title page and in the running heading.

DSMFRONT

The DSMFRONT APF processes the :FRONTM tag. This macro establishes the basic formatting environment for the front matter section of the document including heading definitions and column layout. The DSMFRONT APF performs the following processing:

1. Sets the &@head1 symbol off. This symbol, if defined, is used as a prefix for level one headings. Since prefixes can not be used in the front matter section, the &@head1 symbol is turned off.

2. Redefines heading levels zero through four so they will not be numbered and will not be in the table of contents.

3. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)

4. Calls the DSM#STYL macro to set up the column layout. The front matter section of a general document is always formatted in either one-column style or offset style. If &SYSVARS is "two," indicating that the body of the document will be in two-column format, the front matter is formatted in one-column style. One column style is also used if the body will be in one-column style. If &SYSVARS is "off," offset style will be used for the front matter. See "Starter Set Initialization" on page 19 for details on the DSM#STYL macro.

5. Changes page numbering to roman numeral style.

DSMABSTR

The :ABSTRACT section tag is processed by the DSMABSTR APF. The abstract is part of the front matter of the document. Page layout and heading definitions do not need to be changed because these are governed by the front matter section macro. All the DSMABSTR APF needs to do is to generate a new page with a level one heading of "ABSTRACT" on it. It also resets the &@shead symbol which is used in the running footing. The DSMABSTR APF performs the following processing:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)

2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)

3. Resets &@shead to the value of &LL@Abstr which is "Abstract." The &LL@Abstr symbol is defined in DSM#SETS. The &@shead symbol is used in the running footing.

4. Creates a level one heading for the abstract using the &LL@Abstr symbol defined in DSM#SETS to "Abstract" and the .H1 [Head Level 1] control word.

5. Remaps the paragraph tag to the DSMPARA1 APF. This is because paragraphs following level one headings have a slightly different format than other paragraphs. See "Paragraphs" on page 87 for a complete explanation on why the :P tag is remapped.

DSMPREF

The DSMPREF APF processes the :PREFACE tag. The preface is part of the front matter of the document. Page layout and heading definitions do not need to be changed because these are governed by the front matter section macro. All this macro needs to do is generate a new page with a level one heading of "PREFACE" on it. It also resets the short heading symbol (&@shead) that is used in the running footing.

The DSMPREF APF performs the following processing:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)
3. Resets the &@shead symbol to &LL@Pref which is defined in the DSM#SETS macro to "Preface." &@shead is used in the running footing.
4. Creates a level one heading for the preface using the &LL@Pref symbol which is "Preface" and the .H1 [Head Level 1] control word.
5. Remaps the paragraph tag to the DSMPARA1 APF. This is because paragraphs following level one headings may have a slightly different format than other paragraphs. See "Paragraphs" on page 87 for an explanation on why the :P tag is remapped.

DSMTOC

The DSMTOC APF processes the :TOC tag and produces the table of contents section by performing the following processing:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)
3. Resets the &@shead symbol to &LL@ToC symbol which is defined in the DSM#SETS macro to "Table of Contents." This symbol is used in the running footing and is defined in DSM#SETS
4. Uses the .TC [Table of Contents] control word to format the table of contents. &LL@ToC is used as the title parameter on the .TC [Table of Contents] control word.

DSMFLIST

The :FIGLIST tag is processed by the DSMFLIST APF which produces the List of Illustrations section. The entries for the List of Illustrations are collected in the #FIGLIST macro by the DSMFCAP APF.

On the first pass the List of Illustration section will normally be empty because no figures will have been processed yet. On the second pass the #FIGLIST macro which has been created during the first pass is printed out.

The DSMFLIST performs the following processing:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)

3. Resets the &@shead symbol to &LL@LstIl symbol which is defined in the DSM#SETS macro "List of Illustrations." The &@shead symbol is used in the running footing.

4. Prepares to call the #FIGLIST macro. The #FIGLIST macro is defined one line at a time as each captioned figure is encountered. It is not predefined in the macro library. See "Examples and Figures" on page 115 for more details on how #FIGLIST is created. In preparing the #FIGLIST macro, DSMFLIST:

   a. Defines line number 1 of the #FIGLIST macro to create a level one heading for the list of illustrations page. The entries to create the rest of the page will already be defined in the #FIGLIST macro on the second pass. On the first pass the rest of the #FIGLIST macro is empty as no figures will have been encountered yet.

   b. Saves the formatting environment because the #FIGLIST macro changes the formatting environment and we want to be able to completely restore it.

   c. Restores the control word separator to the default setting (;) with the .DC [Define Character] CW control word. We constructed the #FIGLIST macro as we processed the figures on the first pass through the document and we used the semicolon as the control word separator. Therefore, we have to make sure that it is a semicolon while the #FIGLIST macro is being processed.

5. Calls the #FIGLIST macro to format the list. See the discussion of the :FIGCAP tag in "Examples and Figures" on page 115 to see how the #FIGLIST macro is constructed.

6. Restores the previous formatting environment.

7. Ends the page with a .PA [Page Eject] NOSTART control word.

DSMBODY

The DSMBODY APF processes the :BODY tag. This macro starts a new section, adjusts the style of headings and page numbers, and:

| 1. Clears the &@head symbol.

2. Sets the &@head1 symbol equal to the &@bodyhead1 symbol if the &bodyhead1 symbol was defined in DSMPROF3. The &@head1 symbol is used as a prefix for level one headings in the body.

   The .SE [Set Symbol] control word line for setting &@bodyhead1 is in the profile, but it is commented out. If the user wants to set a prefix for the level one headings in the body, all that needs to be done is remove the comment (.*) and fill in the prefix.

3. Sets up a local symbol if &SYSVARH is not "no" (it means that we're numbering headings). We'll use the symbol on the .DH [Define Head Level] control words that define the headings. If &SYSVARH is "no," we won't set &*a. When we use it on the .DH [Define Head Level] control word line, &*a will resolve to a null (or nothing), which is fine, because the default for .DH [Define Head Level] is NONUM, which is what we want.

4. Puts all level zero through four headings into the table of contents. All but level zero and one headings will be numbered if &*a was set to "num." Level zero headings are never numbered.

5. Tests the value of the &@head1 symbol. The &@head1 symbol may contain a prefix for level one headings in the body of the document. It will have a value of something like "Chapter" or "Part" and will have been set by the user in the profile. If there is a prefix for level one headings (that is, the &@head1 symbol exists), we will want to also number these headings so that they will be labeled "Chapter 1.," "Chapter 2." and so on. When SCRIPT/VS numbers headings it puts the number at the beginning. Since we want the number to appear after the prefix we will have to number these headings ourselves instead of letting SCRIPT/VS do it.

The following line tests for the existence of the &@head1 symbol

```
    .an &E'&@head1 eq 0 .dh 1 num
```

The line shown above, is still logically part of the .IF [If] control word line that is 6 lines above it in the DSMBODY APF. We'll adjust the .DH [Define Head Level] control word for H1 to be numbered only if we're numbering headings (&SYSVARH isn't "no") and we're not prefixing them (&@head1 doesn't exist). In this case, we'll change the H1 back to be numbered and let SCRIPT/VS do the numbering.

6. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
7. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)
8. Calls DSM#STYL to reset page layout to the value of &SYSVARS. This may be one-column, two-column or offset. See "Starter Set Initialization" on page 19 for details on the DSM#STYL macro.
9. Sets page numbering to arabic and resets the page number to 1.

DSMAPPD

The :APPENDIX section tag is processed by the DSMAPPD APF. Aside from starting a new document section, the primary purpose of this macro is to establish serial lettering and prefixing for level one headings. The DSMAPPD APF performs the following processing:

1. Resets the heading counter symbol, using the .GS [GML Services] HCTR control word to start with A.0. This causes the level one heading numbers in the appendix to be letters—A, B, C, and so on.
2. Sets the &@head1 symbol to the value of &LL@Appdx which is "Appendix." This symbol is used as a prefix for level one headings and is defined in DSM#SETS.
3. Adjusts the heading definitions to turn numbering off for level one headings. This is done because the numbering is handled by the DSMHEAD1 APF when headings are prefixed as well as numbered.
4. Resets the definitions for level zero and level two through four headings to ensure that all of these headings go into the table of contents and are numbered or not numbered according to the value of &SYSVARH. If the appendix section follows the body, these redefinitions are not really necessary.

   If heading numbering is on (&SYSVARH isn't "no"), the level two headings will be numbered A.1, A.2, and so on. Level three and four headings will also be lettered and numbered.
5. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
6. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)
7. Calls DSM#STYL to reset page layout to the value of &SYSVARS. This may be one-column, two-column or offset. See "Starter Set Initialization" on page 19 for details on the DSM#STYL macro.
8. Resets page numbering to arabic style. If the appendix section follows the body section, this isn't necessary because page numbers will already be arabic numerals.

DSMBACKM

The DSMBACKM APF processes the :BACKM tag. The primary characteristics of the back matter section are that headings are not numbered and a two column layout is always used regardless of the value of &SYSVARS. The DSMBACKM APF performs the following functions:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Sets the &@head1 symbol off. This symbol is used as a prefix for level one headings. Prefixes may be in effect for either the body of the document or the appendix section, if there is one. In either case, we want to get rid of the prefix.
3. Turns off head level numbering by redefining the headings.
4. Redefines head levels to place level 0 through 4 headings in the table of contents. This is done just in case the previous document section heading definitions did not make entries into the table of contents.
5. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)
6. Calls the DSM#STYL macro to get a two-column layout regardless of the value of &SYSVARS. See "Starter Set Initialization" on page 19 for details on the DSM#STYL macro.

DSMEGDOC

The :EGDOC tag is processed by the DSMEGDOC APF. This APF produces the cross reference listing and writes out the SYSVARW file of IDs.

1. It calls the DSM#XLST macro if this is the last pass (&@lastpass = "yes") and a cross reference listing has been requested (&SYSVARX = "yes"). See "Cross-References" on page 147 for details on how the cross reference listing is produced.
2. If &SYSVARW has been set and this is the last pass, this macro calls DSM#WRIT to write out the file with the cross reference IDs in it. See "Cross-References" on page 147 for details on how the IDs are collected, used, and written out.

Modifications to Document Sections

Adding a Section

Probably the most common change that people make to the document section macros is to add a new document section. In order to do this, you need to first make some decisions.

- What major section of the document does the new section belong in? Front matter, back matter, body, or it is a major section itself?
- What will the tag for it be called?
- Are there any attributes for the tag?
- Should the section be labelled and if so, how? Should it have a standard label like "Abstract" or a variable label that the user supplies?
- Do we want to change the running footing to match the new label?

Let's make some basic assumptions and try it. Suppose we want to add a new section to the front matter called "Summary of Amendments." It will always be labelled that way and we want the running footing to reflect that heading. We'll call the tag :AMEND.

Now we need to do a few things to implement it. First, we have to enable the tag in DSMPROF3. We'll map it to a macro named AMEND. Let's assume that it will have no attributes. So we'll add

```
.aa amend amend (noatt)
```

to DSMPROF3 where all the other tag-to-APF mappings are.

Next we need to write the APF for the :AMEND tag. We must first decide how we want to process it. Let's assume that we want it to be just like the abstract section—a new page (odd if we're duplexing) with a level one heading at the top which says "Summary of Amendments," and a running footing that says the same thing.

```
.dsm#rset
.dsm#dupl
.'se @shead '&LL@Amend
.'h1 &LL@Amend
.aa p dsmpara1
```

Let's look a little closer at what we've done. The DSM#RSET macro will ensure that there are no open text structures, such as lists or figures. The DSM#DUPL macro will end the current page and get us to the beginning of the next page (odd if we're duplexing). These two macros are explained in detail in "Miscellaneous" on page 163.

We've reset the value of &@shead to &LL@Amend. Remember &@shead is used as the text in the running footing. We've got open quotes on &LL@Amend because it will probably contain blanks. We've used the .H1 [Head Level 1] control word to get the level one heading using the standard text we've set up in &LL@Amend. We had to use the control word modifier on both the .SE [Set Symbol] and .H1 [Head Level 1] control words so that we don't need to worry about what's in &LL@Amend (just in case someone changes it and puts a semi-colon in there).

The last line in the macro remaps the :P tag to the DSMPARA1 APF to get the right style of paragraph after the level one heading.

The one remaining thing to do is define the &LL@Amend symbol. We need to set it to "Summary of Amendments" and the logical place to do this is in DSM#SETS where all the other &LL@... symbols are set up.

```
.se LL@Amend  'Summary of Amendments
```

We could have just used "Summary of Amendments" rather than put it into a symbol and then use the symbol. The reason we've done it this way is to facilitate changing it (or translating it).

Changing the Section Label

Let's change one of our assumptions and modify the APF to use the residual text of the :AMEND tag as the label for the page. All we have to do is add a .GS [GML Services] SCAN control word to our macro and use the residual text instead of &LL@Amend. Here's what the macro would look like:

```
    .dsm#rset
    .dsm#dupl
    .gs scan *title
    .'se @shead '&*title
    .'hl &*title
    .aa p dsmparal
```

We also might want to set it up so that if no text was given with the tag, the standard text, &LL@Amend, will be used. Because every tag is considered to have residual text, it's a little difficult to tell if the text really belongs to the tag or not. However, because we are creating a section tag, we could assume that the tag will always be followed immediately by a paragraph tag, or some other text element tag. This is only a moderately risky assumption, but it s already been made for some of the title page tags, such as :DATE.

In this case, we would need to modify our macro to be as follows:

```
    .dsm#rset
    .dsm#dupl
    .gs scan *title
    .'if &L'&*title eq 0 .'se *title '&LL@Amend
    .'se @shead '&*title
    .'hl &*title
    .aa p dsmparal
```

We've tested the length of the residual text that is in the &*title symbol. If there isn't any, it means that another tag was encountered immediately following the :AMEND tag, or that we are at the end of the file. In these cases, we'll set the &*title symbol to the standard section label, &LL@Amend.

Changing the Layout

Another modification we could make is to change the page layout style for one or more of the document sections. Let's keep going with the Summary of Amendments section we just created. Suppose that even though it is in the front matter, which is always formatted with a one-column page layout, we wanted it to always be formatted in two column style. All we need to do is add one line to the AMEND macro we wrote above.

```
    .dsm#styl two
```

This is a call to the DSM#STYL macro. We're also passing it "two" as a parameter which will cause it to set up a two-column format for us. That's all there is to it.

That was simple, wasn't it? However, we've created a problem by doing this. If we assume that the summary of amendments section is in the front matter, we need to be concerned about the page layout style for the other sections in the front matter. If you remember, we said that the abstract and preface sections didn't need to establish their own page layout because the layout was handled by the front matter tag. If we put the summary of amendments before either the abstract or the preface, we will get the wrong page layout for the section that follows it.

There are several ways to handle this. We could put the summary of amendments before the front matter, but that wouldn't make much sense, because it really is part of the front matter. We could let the abstract section and the preface section handle their own page layout by calling the DSM#STYL macro in the same manner that the front matter did.

Another approach would be to define an end tag for the summary of amendments section that would reset the page layout.

Either of these last two methods will work fairly well. The best approach would be to let each section set up its own page layout and remove the page layout lines from the DSMFRONT APF. The reason this is better is that having different page layouts in the front matter means that the page layout is no longer a function of front matter. It is a function of the section. This would logically lead us to put the page layout in the each section rather than in the front matter.

Let's work through creating the end tag solution, just for practice. First we will need to add the name of the APF for the end-tag to the .AA [Associate APF] control word line we added to DSMPROF3.

```
.aa amend amend (noatt) eamend
```

Next we need to write the EAMEND APF. All this macro must do is restore the page layout. The layout for the other sections in the front matter depends on the value of &SYSVARS. If &SYSVARS is two, we will set up a one-column layout. Otherwise, we'll use the actual value of &SYSVARS, which will be either "one" or "off." Here's what the EAMEND APF would look like:

```
.if &SYSVARS eq two .dsm#styl one
.el .dsm#styl
```

Changing the Appendix Headings

Another thing we could do is to change the way level one headings in the appendix are labelled. They are prefixed with "Appendix A:," "Appendix B:," and so on. There are a couple of things we could do here. Suppose we wanted them numbered instead of labelled. All we would have to do is change the .GS [GML Services] HCTR control word line in the DSMAPPD APF.

```
.gs hctr 1.0
```

Suppose we wanted to change the prefix from "Appendix:" to "Supplement." If you look back at the DSMAPPD APF, you'll see that we've used the LL@Appdx symbol instead of the word "Appendix." We can either change the value of &LL@Appdx in the DSM#SETS macro to "Supplement" or we can change the .H1 [Head Level 1] control word line. To accomplish the task using the first method, we need to change

```
.se LL@Appdx 'Appendix
```

to

```
.se LL@Appdx 'Supplement
```

in the DSM#SETS macro.

However, now we've got a symbol (&LL@Appdx) whose name does not accurately reflect its value. It would be cleaner to add the name &LL@Supp and change the references in the DSMAPPD APF from &LL@Appdx to &LL@Supp.

Changing the Table of Contents Format

The starter set formats all of the entries in the table of contents in the same size font for page printers. The entries are all generated automatically by SCRIPT/VS from the heading control words. The fonts to use for the entries are specified in DSMPROF3 on the .DH [Define Head Level] control words. The definitions for the fonts used are shown in Figure 6 on page 25.

It would be a simple matter to put different level headings in different size fonts for page printers by just changing the font definitions. Although this works in terms of producing output, there is a major problem with the way it looks. The table of contents entries are formatted using an internally generated .SX [Split Text] control words which do not allow font changes. So if the entry were in a larger font, the page number on the right side would also be in the larger font, as well as the dot leader. Unfortunately this looks very strange. The numbers and leader dots do not line up very well and the result is very unappealing.

In order to create a table of contents using different size fonts for page printers several changes need to be made to the macro library including generating the entry ourselves using the .PT [Put Table of Contents] control word. Let's assume that there are no level zero headings in the document and the only ones we are interested in changing are the level one headings.

First a larger font needs to be defined. The font definition is in DSMPROF3:

```
 .df hd1toc type(10 bold) up
```

Let's change this to:

```
  .df hd1toc type(16 bold)
```

The next step is to *not* have SCRIPT/VS automatically generate the entry. This means changing the .DH [Define Head Level] control word that is also in DSMPROF3. Actually there are 2 .DH [Define Head Level] control words for level one headings because all of the parameters wouldn't fit on one line. These are:

```
.dh 1 nus nohy nup font hd1 &*n spaf &@h1sp pa left sect tfont hd1toc ts
.dh 1 spbf &@hspbf
```

We'll keep SCRIPT/VS from generating the entry by adding NTC to the second one.

```
.dh 1 spbf &@hspbf ntc
```

Now, that gets it set up correctly to start with, however, which headings go in the table of contents changes as we go through the document. Headings in the front matter and the back matter do not go in the table of contents. Headings in the body and the appendix do go in the table of contents. The heading definitions are changed accordingly for each of these document sections. (See Figure 13 on page 77.)

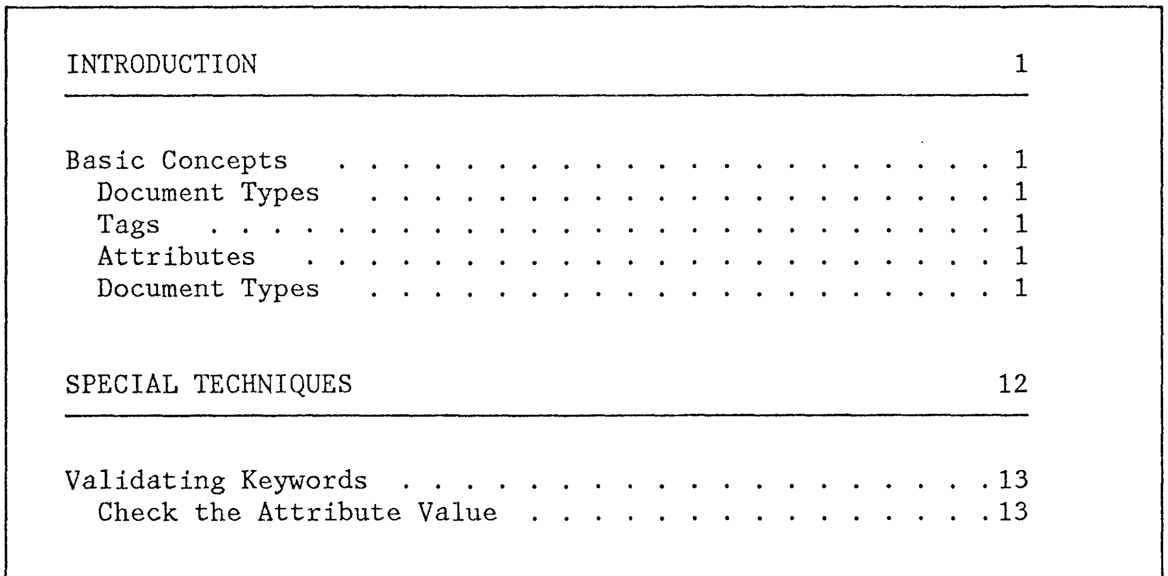
Therefore, we need to adjust the DSMBODY and DSMAPPD APFs to not change the level one headings back to TC after we've changed them to NTC. In both APFs the line that reads:

```
 .dh 1 tc nonum
```

needs to be changed to

```
 .dh 1 ntc nonum
```

Now that we've stopped SCRIPT/VS from generating the entries, we will need to generate the entries ourselves in the DSMHEAD1 APF. We will use a series of .PT [Put Table of Contents] control words to put the appropriate control words and text into the table of contents file. Figure 12 on page 67 shows the results we want.



```
INTRODUCTION                                            1
_________________________________________________________

Basic Concepts . . . . . . . . . . . . . . . . . . . . . 1
  Document Types . . . . . . . . . . . . . . . . . . . . 1
  Tags . . . . . . . . . . . . . . . . . . . . . . . . . 1
  Attributes . . . . . . . . . . . . . . . . . . . . . . 1
  Document Types . . . . . . . . . . . . . . . . . . . . 1


SPECIAL TECHNIQUES                                     12
_________________________________________________________

Validating Keywords . . . . . . . . . . . . . . . . . .13
  Check the Attribute Value . . . . . . . . . . . . . .13
```

Figure 12. Sample Output: The format of the table of contents can be changed to look like this by generating the table of contents entries in the APF for the :H1 macro.

The following lines need to be added to the DSMHEAD1 APF:

```
.'h1 &@head
.se hdnum = &
.pt .sp 2
.pt .tp &dh'&$cl.dh right
.pt .bf hd1toc
.pt .li &@head.&$TAB
.pt .pf
.pt .bf hd2toc
.pt .ct &hdnum
.pt .hr left to right
.pt .sp 1
.pt .pf
```

These lines save the current page number in &hdnum and put the following lines into the table of contents file:

1. Space 2 lines.
2. Set a right aligned tab at the right side of the page. Notice that this implementation uses the tab rack. The starter set typically avoids using the tab rack whenever possible so as not to interfere with any possible user tabs that have been defined.
3. Begin the font for level one entries.
4. Format the heading text following by a tab character in literal mode.[24]
5. Restore the previous font.

[24] The .LI [Literal] control word is necessary here because if the contents of the .PT [Put Table of Contents] control word is simple text it will automatically generate a .SX [Split Text] control word for it and we don't want that. By starting the line with a control word we get the line put in exactly as we coded it.

6. Start the font that will be used for the level two headings in the table of contents (hd2toc). We do this because we want to format the number in the same font as the rest of the entries in the table.
7. Format the page number that is in &hdnum preceding it with a continuation character. We also used literal mode here to avoid the .SX [Split Text] control word from being built by the .PT [Put Table of Contents] control word. The &$CONT causes the line to be treated as a line of text by .LI [Literal]
8. Draw a horizontal rule from the left to the right to provide extra visual separation.
9. Space 1 line.
10. Restore the previous font.

Creating a Table of Contents For Each Chapter

For some applications it is more appropriate to provide a table of contents for each chapter rather than a single table of contents at the beginning of the document. The table of contents entries would need to be collected during the first pass for each chapter of the book. On the second pass the table of contents would need to be produced by the DSMHEAD1 APF.

While this is fairly easy to accomplish it does present one problem which is not easy to solve. Since the table of contents for the chapter is inline after the heading, and since it is empty on the first pass, text will shift on the second pass. This means that the page numbers from the first pass will most likely be wrong on the second pass if we don't reserve the correct amount of room for the partial table of contents. We have no good way of guessing how much space to allow for the table of contents on the first pass. The best we can do is use SYSVAR 'W' to save the table of contents entries from one formatting run and use that as input to the next run.

Preparation

Let's assume that only level 2 headings are going to be put into this partial table of contents and that we won't disable the main table of contents. This means that we will need to modify the DSMHEAD1 APF and the DSMHEAD2 APF. We'll collect the level 2 entries in a separate array for each chapter (level one heading). Therefore we'll need to create a unique array name for each chapter. As we will see below, we're also going to need a unique counter for each head 2. To do this we'll initialize some counters to zero in the profile.

```
    .se @hd1ctr = 0
    .se @hd2ctr = 0
```

and we'll increment the &@hd1ctr symbol each time we encounter a head 1. This means adding

```
    .se @hd1ctr = &@hd1ctr + 1
```

to the DSMHEAD1 APF right after the .H1 [Head Level 1] control word. Now we can use the counter in the array name so we'll get a unique array for each chapter.

Saving the Table of Contents Information

Our next problem is how to save the entries in the array. The approach that we've taken here is to save the information in an array in the form of .SX [Split Text] control words. The name of the array will be keyed off of the level 1 heading number. For example, the entries for chapter 1 will be put in an array named &toc1, and the entries for chapter 2 will be put in an array named &toc2, and so on. The heading number (&@hd1ctr) symbol is used to construct the name of the array.

We'll have to add some logic to the DSMHEAD2 APF to save the text of the heading and the page number in the appropriate array. The best approach is probably to set up .SX [Split Text] control word lines in the array and then we can just dump out the array when it's time to produce the table of contents. The left hand side of the split text control will be the text of the heading which we have in the &@head symbol. The middle part will be a period and a blank to produce a dot leader. The right hand part needs to be the page number.

However, simply including the page number symbol (&) in the split text line won't work because SCRIPT/VS won't resolve it as a page number for us. What we are going to have to do is create a unique symbol name to contain the page number for each level 2 heading and then use this symbol in the split text line. That's where the unique symbol name for each head 2 comes in. The symbol that contains the page number for the head 2 needs to be unique because it has to survive until it's time to dump out the table of contents.

To do all this we'll add:

```
    .se @hd2ctr = &@hd2ctr + 1
    .se *sx '.sx f /&@head./ ./&PG&@hd1ctr.&@hd2ctr../
    .se toc&@hd1ctr.(&@hd2ctr.) '&*sx
    .se PG&@hd1ctr.&@hd2ctr = &
    .se curr&@hd1ctr = &curr&@hd1ctr + 1
```

to the end of the DSMHEAD2 APF. Each time we come through the DSMHEAD2 APF we'll increment the &@hd2ctr symbol by one to get a unique number. We'll use the &PG&@hd1ctr.&@hd2ctr symbol to represent the page number and use this symbol in the .SX [Split Text] line. Then we'll set the &PG&@hd1ctr.&@hd2ctr symbol to be the page number. We have to set this symbol after we use it in order to get the unresolved symbol name into the .SX [Split Text] text. If the heading shifts to a different page due to widow zone or keep processing we would have picked up the wrong page number if we had set &PG&@hd1ctr.&@hd1ctr before we used it. This way the .SX text will contain the symbol name which will resolve to the correct value of &PG&@hd1ctr.&@hd2ctr when we use the array.

There's one more thing we need to do in the DSMHEAD2 APF. We need to keep track of how many entries we've put into the array. When we come around for the second pass the array elements will still exist but the element counter (0) will have been set back to zero for the second pass. In order to dump out the array, we will need to reset element zero of the array, so we need to know how many elements we put into it. We'll use the &curr&@hd1ctr symbol for this and increment it each time we put something into the array.

The delimiters we've used in the .SX [Split Text] line are a problem. Slashes won't work if there happen to be any slashes in the text of the heading. Therefore, we really should use some obscure hexadecimal number as the delimiters instead. For example hexadecimal 1's,

```
    .se *sx '.sx f &X'01.&@head.&X'01. .&X'01.&PG&@hd1ctr.&@hd2ctr..&X'01.
```

This way we don't need to worry about a character in the heading text being interpreted as a delimiter on the .SX [Split Text] control word.

Producing the Partial Table of Contents

The next step is get the DSMHEAD1 APF to produce the table of contents. We'll need to format the heading for the table of contents. Let's use highlight font 2 to do this. We need to add the following lines to the DSMHEAD1 APF right after the heading is generating with the .H1 [Head Level 1] control word:

```
   .bf hi2
   .sx /&LL@ToC.//&LL@Page./
   .pf
   .sp
```

We didn't leave any extra space at the top because there is already space after the heading. We did leave a space after the table of contents heading, just to make it look nice.

The next step is to check if there are any entries in the array before we bother to dump it. The &curr&@hd1ctr symbol is used to count entries. If it doesn't exist we want to set it to zero.

```
  .if &E'&curr&@hd1ctr eq 0 .se curr&@hd1ctr = 0
```

Then, if it is zero, there is nothing to dump so we can skip to the end of the lines we are adding.

```
   .if &curr&@hd1ctr eq 0 .go next
```

If the value of &curr&@hd1ctr is not zero, it means that there are entries to be formatted. We will need to transfer the number of entries from &curr&@hd1ctr to element zero of the array. Again, since element zero of the array is cleared for the second pass we need to reconstruct it in order to print the array.

```
   .if &toc&@hd1ctr.(0) eq 0 .se toc&@hd1ctr.(0) = &curr&@hd1ctr
```

The next step is to set the array separator to be a .BR [Break] control word and then dump the array out.

```
    .dc asep & a .
    .'se a = ';br;'
    &toc&@hd1ctr(*)
```

Since we are incrementing &curr&@hd1ctr for each level 2 heading and since it maintains its value from the first to the second pass we have to initialize it to zero after we have used it.

```
    .se curr&@hd1ctr = 0
```

The last step is to reset the &@hd2ctr symbol to zero to start numbering level two headings again with 1 in the next chapter.

```
   ...next
   .se @hd2ctr = 0
```

The New Macros

The new DSMHEAD1 APF looks like this after our modifications:

```
.*  DSMHEAD1: Tag = H1  Attr = ID, STITLE  Format level 1 heading.    *
.*  Advances to next/odd page.  Head1's are numbered in the body if    *
.*  either head level numbering is on or &@head1 exists.               *
.*  Reset any open lists, etc.                                         *
.**********************************************************************
.dsm#rset H.-1
.dsm#dupl
.gs scan @head
.*           PREFIX HEADING WITH &@head1, IF IT EXISTS                 *
.if &E'&@head1 eq 1 .gs hctr 1
.'th .'se @head '&@head1 &@xref(1).. &@head
.*          SET &@shead FOR THE RUNNING FOOTING TO HEADING OR STITLE   *
.'se @shead '&@head
.gs exatt stitle as dsm@shd
.* CREATE THE HEADING, PROCESS THE ID AND DON'T INDENT 1ST PARAGRAPH   *
.'h1 &@head
.*  FORMAT THE PARTIAL TABLE OF CONTENTS
.se @hd1ctr = &@hd1ctr + 1
.bf hi2
.sx /&LL@ToC.//&LL@Page./
.pf
.sp
.if &E'&curr&@hd1ctr eq 0 .se curr&@hd1ctr = 0
.if &curr&@hd1ctr eq 0 .go next
.if &toc&@hd1ctr.(0) eq 0 .se toc&@hd1ctr.(0) = &curr&@hd1ctr
.dc asep & a .
.'se a = ';.br;'
&toc&@hd1ctr.(*)
.dc asep
.se curr&@hd1ctr = 0
...next
.se @hd2ctr = 0
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara1
```

The new DSMHEAD2 APF looks like this after our modifications:

```
.* DSMHEAD2: Tag = H2   Attr = ID  Formats level 2 heading            *
.* Resets any open lists, etc.                                        *
.*********************************************************************
.dsm#rset H.-2
.gs scan @head
.* CREATE THE LEVEL 2 HEADING, PROCESS THE ID & DON'T INDENT 1ST PARA *
&@rc1
.'h2 &@head
&@rc2
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara2
.se @hd2ctr = &@hd2ctr + 1
.se *w ='.sx f &X'01.&@head.&X'01. .&X'01.&PG.&@hd1ctr&@hd2ctr..&X'01.
.se toc&@hd1ctr.(&@hd2ctr.) '&*w.
.se PG&@hd1ctr.&@hd2ctr = &
.se curr&@hd1ctr = &curr&@hd1ctr + 1
```

Using SYSVAR 'W'

The modifications outlined above will work except that no space will be reserved for the partial table of contents on the first pass. This means that on the second pass when the table is formatted, subsequent text will shift. This will, potentially, make the page numbers that we have saved for the headings wrong.

The solution suggested here is to save the table of contents information in the SYSVAR 'W' file and use it on subsequent runs. It important that the TWOPASS option always be used so that the information in the SYSVAR 'W' file will be correct. This way the table will be filled in with information from the previous run on the first pass. This is a fairly good way to approximate the size of the real table which will be formatted on the second pass.

This means adding the following lines to the end of the DSM#WRIT macro:

```
.*   SAVE THE PARTIAL TABLE OF CONTENTS INFORMATION FOR EACH HEADING
.se *head = 1
...outer
.wf .se curr&*head = &curr&*head
.se *elem = 1
...inner
.wf .'se toc&*head.(&*elem.) = '&toc&*head.(&*elem.)
.se *elem = &*elem + 1
.if &*elem le &toc&*head.(0) .go inner
.se *head = &*head + 1
.if &*head le &@hd1ctr .go outer
```

These lines loop through all of the level 1 headings (up to the current value of &@hd1ctr) the contents of the &curr&@hd1ctr symbol. It also loops through the elements of the table of contents arrays writing out .SE [Set Symbol] control word lines to set each element of the array.

We've made this modification for this chapter. The partial table of contents looks like this:

Table of Contents	**Page**
Overview	57
Document Section Macros	57
Modifications to Document Sections	62

That's all there is to it!

Headings

Overview

The APFs that process the heading tags (DSMHEAD0-DSMHEAD6) perform formatting functions not available on the .DH [Define Head Level] control word and then invoke the appropriate heading control word. These functions include such things as:

- Revision code placement relative to headings
- Setting symbols for text of running headings and footings
- Making sure no lists, figures, footnotes and so on are currently in progress
- Handling ID attributes for cross references.

The head level definitions, however, are sprinkled throughout the APFs and are frequently changed based on what document section we are in. Cross-referencing for headings, as well as the ID attribute processing, are discussed separately in "Cross-References" on page 147.

Head Level Definition

The head level definitions originate in the defaults established within SCRIPT/VS itself. Some, such as "DOT" are never overridden by the starter set. Others such as "FONT" and "TFONT" which originally are set to the default font by SCRIPT/VS, are changed just once in DSMPROF3 during initialization. For page printers, various sizes and styles of fonts have been selected for the different heading levels both in the body of the document and in the table of contents. These are described in "Starter Set Initialization" on page 19 in Figure 6 on page 25. :H5 and : H6 are not listed in the table of contents. :H4 headings will be listed in the table of contents only in the body of the document (marked with :BODY.)

Prefixing Level One Headings

Level 1 headings in the body of the document can be prefixed by activating a .SE [Set Symbol] control word line that is in DSMPROF3.

```
.*.se @bodyhead1 'Chapter
```

This symbol, &@bodyhead1, when set will cause level 1 headings to be prefixed with the word "Chapter" when a :BODY tag is used. In other words, to prefix level one headings you need to:

1. Set &@bodyhead1 to the prefix
2. Use a :BODY tag.

These prefixed headings will automatically be numbered regardless of the value of &SYSVARH. The level one headings in the appendix are automatically prefixed and numbered.

Numbering and prefixing is done through the following processing sequence:

- The &@bodyhead1 symbol is set in the profile to the value of prefix ("Chapter" or "Part" for example).
- The &@head1 symbol is defined to be "off" by DSMPROF3.
- The DSMBODY APF transfers the value of &@bodyhead1 to &@head1 if &@bodyhead1 exists.
- The DSMHEAD1 APF will number the level one headings and build the heading text from the prefix (&@head1), the number, and the residual text of the :H1 tag.

Prefixing is specifically turned off in the back matter and the front matter by the DSMBACKM and DSMFRONT APFs. It is also always turned on and set to "Appendix" for the appendix section by the DSMAPPD APF.

Head Level Numbering

Numbering of head levels may appear somewhat confusing at first glance at the APFs. Numbering can be requested by using the SYSVAR 'H' option of the SCRIPT/VS command. However, numbering is specifically turned off by most of the section tags (:FRONTM, :BACKM, and :APPENDIX).

For level one headings numbering is manipulated depending on whether or not the user has specified a prefix for these headings. (See "Prefixing Level One Headings" on page 75 for more details on prefixing.) If a prefix has been given, numbering is handled externally in the DSMHEAD1 APF for level one headings. Otherwise, it is handled automatically by the .H1 [Head Level 1] control word. Since level one headings in the appendix are prefixed with the word "Appendix," they are always numbered and the numbering is done by the DSMHEAD1 APF. Level 2 through 4 headings in the appendix are numbered only if &SYSVARH is "yes." Level 5 and 6 headings are never numbered.

See Figure 13 on page 77 for details on where heading numbering is turned off and on.

Revision Codes for Headings

Revision code placement is controlled with the .RC [Revision Code] ADJUST control word. The default location is two characters to the left of the column and this is used for one- and two-column formats in the starter set. However, in offset style, the revision code location is changed by the DSM#STYL macro to be approximately 15 characters to the left of the column margin. This works fine for revision codes around text, however, it presents a problem for revision codes around headings.

Level 0 through 4 headings in offset style cause a section break which formats them at the left margin of the page. A revision code placed 15 characters to the left of the heading falls off the page. This is allowed to happen for level zero and level one headings.[25] For level two through four headings, the heading APFs adjust the revision code location to be relative to the heading just before issuing the heading control word. After the control word is processed, the revision code location is adjusted back to be relative to the column text again.

These adjustments are accomplished by setting up two symbols, &@rc1 and &@rc2, whose value depends on the column layout. These are defined in the DSM#STYL macro. For one- and two-column format, the &@rc1 and &@rc2 symbols are both null, in which case they do noth-

[25] Since heading levels zero and one are sometimes right aligned, the revision code is not adjusted to appear as with heading levels two through four. This is done simply for aesthetic reasons because the revisions would look odd on the left when the heading was right aligned.

```
DSM#STYL Macro
   one-column     - if duplexing, level 0 is aligned outside
   & two-column     if not duplexing, level 1 is aligned outside

   offset         - levels 2 - 4 cause section breaks (SECT)
                  - levels 0 and 1 is left aligned

DSMFRONT Macro
  level 0-4       - do not appear in the table of contents (NTC)
  level 1-4       - are not numbered (NONUM)

DSMBODY Macro
  level 0-4       - do appear in the table of contents (TC)
  level 1         - are numbered (NUM) if not prefixing
                    and &SYSVARH = yes
                  - are not numbered (NONUM) if not prefixing
                    or &SYSVARH = no
  level 2-4       - are numbered (NUM) only if &SYSVARH = yes
                    otherwise they are not numbered (NONUM)

DSMBACKM Macro
  level 0 - 4     - do appear in the table of contents (TC)
  level 1 - 4     - are not numbered (NONUM)

DSMAPPD Macro
  level 0 - 4     - do appear in the table of contents (TC)
  level 1         - are not numbered (NONUM)***
  level 2 - 4     - are numbered (NUM) only if &SYSVARH = yes
```

Figure 13. Heading Definitions: This figure lists all the macros that change the heading definitions using the .DH [Define Head Level] control word and details what changes are made in each case. The primary definitions for headings are given in DSMPROF3, which is not listed here. Additional heading formatting is controlled by the default settings for headings in SCRIPT/VS.

*** Note, when head level ones are prefixed, numbering is done by the DSMHEAD1 APF instead of by SCRIPT/VS so the NONUM option is specified even though the headings are numbered.

ing. For offset style, these two symbols have a value of ".RC ADJUST" and ".RC ADJUST n," where "n" is approximately 15.

Heading Macros

The document section macros and profile described below perform processing which alters the formatting of the headings. The APFs for the heading tags and attributes are also described below. :#np.

DSMPROF3

DSMPROF3 initializes the heading definitions and some symbols which control heading processing. DSMPROF3 performs the following processing relevant to headings:

1. Defines all the fonts for all output devices to be used for headings and for table of contents entries. See Figure 6 on page 25 for details on what these font definitions are.

2. Sets the &@head1 symbol to "off." This symbol is the prefix for level one headings. A :BODY tag is required in order to activate the prefixing of level one headings. The :APPENDIX tag also uses this symbol to get each level one heading labelled "Appendix."

3. Defines several symbols which are used as the values of the SKBF, SPBF and SPAF parameters on the .DH [Define Head Level] control words. These are defined initially with the values for line devices and then are redefined for the page printers.

   The symbols and values for skips and spaces around the headings are shown in the table below:

Symbol Name	Line Devices	Page Devices	Description
@hspbf	0	1.3i	SPBF headings 0-1
@h0sp	5	p14	SPAF heading 0
@h1sp	3	p14	SPAF heading 1
@h2sk	3	p20	SKBF heading 2
@h2sp	2	p11	SPAF heading 2
@h3sk	3	p18	SKBF heading 3
@h3sp	2	p11	SPAF heading 3
@h4sk	3	p14	SKBF heading 4
@h4sp	2	p11	SPAF heading 4

Figure 14. Spacing Symbol Definitions for Headings

4. Defines level 0 and 1 headings to be left aligned. These headings will be changed to be outside aligned if duplexing is in effect (&SYSVARD is "yes") and the column layout style is one- or two-column. All other headings are left aligned.

5. Performs several actions, if heading numbers have been requested with SYSVAR 'H'. Head levels one through four are defined to be numbered. The heading counter is set to the value of &SYSVARH.

6. Defines all headings not to be underscored or uppercase. However, because a heading font is associated with each heading, underscoring and uppercasing may occur as a result of the font specification rather than the head level definition. See Figure 13 on page 77 for heading definitions. See Figure 6 on page 25 for font definitions.

7. All headings are also defined to not be hyphenated using the NOHY parameter on the .DH [Define Head Level] control word.

DSM#STYL

The DSM#STYL macro resets some of the heading definitions according to formatting style (one- or two- column). The DSM#STYL macro performs the following processing relevant to headings:

1. Defines head levels two through four to cause a section break for offset style. Heading level zero is outside aligned for one and two column formats if duplexing is active.

2. Sets the &@rc1 and &@rc2 symbols to null for one- and two-column layouts. These symbols are set to ".RC ADJUST" and ".RC ADJUST n" for offset style. See "Revision Codes for Headings" on page 76 for more details.

DSMAPPD

The DSMAPPD APF resets some of the heading definitions as part of starting the appendix section of the document.

The DSMAPPD APF performs the following processing for headings:

1. Sets the &@head1 symbol to &LL@Appdx, which is "APPENDIX," to cause level one headings in the appendix to be prefixed with the word "APPENDIX." See "Prefixing Level One Headings" on page 75 for more an overview of prefixing headings.
2. See Figure 13 on page 77 for changes made to heading definitions.

DSMBACKM

The DSMBACKM APF resets some of the heading definitions as part of starting the back matter section of the document. The DSMBACKM APF performs the following processing relevant to headings:

1. Sets the &@head1 symbol off to clear any level one heading prefixes that might have existed. Level 1 headings may be prefixed only in the body and the appendix. See "Prefixing Level One Headings" on page 75 for an overview of prefixing headings.
2. See Figure 13 on page 77 for changes made to heading definitions.

DSMBODY

The DSMBODY APF resets some of the heading definitions as part of starting the body section of the document. The DSMBODY APF performs the following processing relevant to headings:

1. Sets the symbol &@head1 to the value of &@bodyhead1. The &@bodyhead1 may have been set in the profile to provide a prefix for level one headings in the body. If the &@head1 symbol is not null, the DSMHEAD1 APF will prefix its value to the heading text and number the headings even if numbering is not on. See "Prefixing Level One Headings" on page 75 for an overview of prefixing headings.
2. Resets level one headings to not be numbered if the &@head1 symbol exists. This is because the numbering is handled by the DSMHEAD1 APF for level one headings if a prefix exists.
3. Resets level two to four headings to be numbered if numbering is on (&SYSVARH = "yes"). This is necessary because if there was a :FRONTM tag, the headings are defined to not be numbered. If head level numbering has been requested with SYSVAR 'H', only headings in the body are numbered.[26]
4. See Figure 13 on page 77 for changes made to the heading definitions.

DSMFRONT

The DSMFRONT APF resets some of the heading definitions as part of starting the front matter section of the document. The DSMFRONT APF performs the following processing relevant to headings:

[26] Head level ones in the appendix are also numbered because they are prefixed with the word "Appendix."

1. Sets the &@head1 symbol off to clear any level one heading prefixes that might have existed. Level 1 headings may be prefixed only in the body and the appendix.

2. See Figure 13 on page 77 for changes made to heading definitions.

DSMHEAD0

The :H0 tag is processed by the DSMHEAD0 APF which formats level zero headings in the following manner:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)

2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)

3. Gets the residual text into the &@head symbol.

4. Puts the text of the heading into the &@shead symbol which is used in the running footing.

5. Processes the STITLE attribute with the DSM@SHD macro. If STITLE was specified, its value becomes the value of &@shead. This will change the text that is formatted in the running footing.

6. Uses the .H0 [Head Level 0] control word to format the heading. The control word modifier is used here because we have no way of determining which characters might be in the text of the heading.

7. Sets the &@tg symbol to "h" to indicate to the DSM@IDS macro that a heading id is being processed and then calls DSM@IDS to process the ID attribute if the ID attribute is specified.

8. Remaps the :P tag to the DSMPARA1 APF. This permits special processing for the first paragraph following a level zero heading. See "Paragraphs" on page 87 for more details about paragraph formatting.

DSMHEAD1

The :H1 tag is processed by the DSMHEAD1 APF which formats level one headings in the following manner:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)

2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)

3. Gets the residual text into the &@head symbol.

4. Tests the existence of the &@head1 symbol. It will exist if there is a prefix for level one headings. It is set either in the DSMBODY APF if the &@bodyhead1 symbol has been defined or in the DSMAPPD APF. The &@bodyhead1 symbol may have been defined in DSMPROF3 and is a prefix for level one headings.

   If &@head1 exists, the heading number is incremented by 1 and the numbering of level one headings is handled here. Otherwise, numbering is handled automatically by the .H1 [Head Level 1] control word. The &@head symbol is reset to contain the prefix, the number and the heading text.

5. Puts the text of the heading into &@shead for use in the running footing, just in case no short title (STITLE) attribute was specified.

6. Calls the DSM@SHD macro to save the short title in &@shead. if a short title (STITLE) attribute was specified. This will change the text of the running footing.
7. Uses the .H1 [Head Level 1] control word to process the heading. The control word modifier is used here because we have no way of determining which characters might be in the text of the heading.
8. Sets the &@tg symbol to "h" to indicate to DSM@IDS that a heading id is being processed and then calls DSM@IDS to process the ID attribute if the ID attribute is specified.
9. Remaps the :P tag to the DSMPARA1 APF. This permits special processing for the first paragraph following a level one heading. See "Paragraphs" on page 87 for more details about paragraph formatting.

DSM@SHD

The DSM@SHD macro processes the STITLE attribute. All it does is save the attribute value in the &@shead symbol which is used in the running footing.

DSMHEAD2

The :H2 tag is processed by the DSMHEAD2 APF which formats level two headings in the following manner:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Gets the residual text in the &@head symbol.
3. Adjusts the location of the revision code. If offset style is being used, &@rc1 will be .RC ADJUST otherwise it has a null value which does nothing. See "Revision Codes for Headings" on page 76 for a full explanation of revision code adjustment around headings.
4. Uses the .H2 [Head Level 2] control word to process the heading text that is in &@head. The control word modifier is used here because we have no way of determining which characters might be in the text of the heading.
5. Adjusts the revision code back to be placed relative to the column text. In number 3. above we had moved the revision code to be relative to the heading text for offset style. Now we are moving it back to be relative to the text. In offset style the &@rc2 symbol has a value of .RC ADJUST n where n is approximately 15. For one- and two-column style, &@rc2 is null.
6. Sets the &@tg symbol to "h" to indicate to DSM@IDS that a heading id is being processed and then calls DSM@IDS to process the ID attribute if the ID attribute is specified.
7. Remaps the :P tag to the DSMPARA2 APF. This permits special processing for the first paragraph following a level 2 heading. See "Paragraphs" on page 87 for more details about paragraph formatting.

DSMHEAD3

The :H3 tag is processed by the DSMHEAD3 APF. The processing in this macro is exactly the same as in the DSMHEAD2 APF except that the .H3 [Head Level 3] control word is used to format the heading instead of the .H2 [Head Level 2] control word.

DSMHEAD4

The :H4 tag is processed by the DSMHEAD4 APF. The processing in this macro is exactly the same as in the DSMHEAD2 APF except that the .H4 [Head Level 4] control word is used to format the heading instead of the .H2 [Head Level 2] control word.

DSMHEAD5

The :H5 tag is processed by the DSMHEAD5 APF which formats level five headings. These headings are in-line with the text that follows and are produced with the following processing:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Gets the residual text into the &@head symbol.
3. Uses the .H5 [Head Level 5] control word to process the heading. The control word modifier is used here because we have no way of determining which characters might be in the text of the heading.
4. Sets the &@tg symbol to "h" to indicate to DSM@IDS that a heading id is being process and then calls DSM@IDS to process the ID attribute if the ID attribute is specified.
5. Sets the &@h5line symbol to the page number followed by a slash, followed by the line counter value. This symbol will be used in the paragraph APF to determine if any text has been processed between the :H5 tag and the next :P tag. Whether or not the :H5 is followed immediately with a :P tag is significant because the :P tag will supply an ending colon to the heading text only if there is no intervening text between the two tags.
6. Saves the heading level number (5) in the &@para5@fnt symbol. This symbols will be used in the DSMPARA5 APF to restart the appropriate font. See "DSMPARA5" on page 89 for how and why this is necessary.
7. Remaps the :P tag to the DSMPARA5 APF. This permits special processing for the first paragraph following a level five heading. See "Paragraphs" on page 87 for more details about paragraph formatting.

DSMHEAD6

The :H6 tag is processed by the DSMHEAD6 APF which formats level six headings. These headings are in-line with the text that follows.

The processing in this macro is exactly the same as in the DSMHEAD5 APF except that the .H6 [Head Level 6] control word is used to format the heading instead of the .H5 [Head Level 5] control word and the &@para5@fnt symbol is set to "6" instead of "5."

Modifications to Headings

Many things regarding headings can be modified simply by changing the heading definition (.DH [Define Head Level]). Since there are several sets of heading definition control words in the starter set macros, you will need to be careful as to where you change the definition. Figure 13 on page 77 shows which macros contain the .DH [Define Head Level] control words and what changes are made in each case. For example, to manipulate the numbering of headings you need to consider the fact that the NUM/NONUM parameters are reset in DSMPROF3, the DSMFRONT APF, the DSMBODY APF, the DSMBACKM APF and the DSMAPPD APF. Depending on the nature of the change you want to make, you might have to modify the definitions in several places.

Capturing Heading Numbers

The numbers that are used for headings are generated by SCRIPT/VS. These numbers can be obtained from the &@xref symbol array for use in running headings and footings. &@xref is a symbol array where the first element of the array contains the number that was used for the last level one heading.[27] The second element of the array contains the number of the last level two headings and so on.

Heading numbers are included in the text of the running footing only if headings are also being prefixed. However, suppose that you wanted to always include the heading number in the running footing even when level one headings aren't being prefixed. This requires capturing the number as well as the text of the heading.

When headings are being prefixed (the &@head1 symbol contains the prefix), they are numbered by the DSMHEAD1 number rather than by SCRIPT/VS.

```
.if &E'&@head1 eq 1 .gs hctr 1
.'th .'se @head '&@head1 &@xref(1).. &@head
```

The DSMHEAD1 APF puts the text of the heading into the &@shead symbol. The lines below show how this is done:

```
.'se @shead '&@head
```

The prefix, the number and the text of the headings are all put into the &@shead symbol and will show up in the running footing.

Since the &@xref symbol is not normally incremented until after the heading has been produced with the .H1 [Head Level 1] control word, we need to capture the number after the heading. However, the .H1 [Head Level 1] control word starts the page which causes the running footing to be formatted. Therefore, in order to get the heading number into the footing we need to both capture the number and reformat the running footing. The number can be captured in the same manner that is shown above when prefixing is being done by adding the following lines to the DSMHEAD1 APF:

```
.'h1 &@head
.if &E'&@head1 eq 0 .an &SYSVARH ne no
.'th .'se @shead '&@xref(1).. &@shead
.th .rf execute
```

We check first that prefixing is not being done, because if it is we don't need to do anything because the running footing will already contain the number. We use just the first element of the &@xref array because we want only the level one heading number to show up in the footing.

An even simpler method of accomplishing the same task can be done by forcing the headings to be numbered by the DSMHEAD1 APF. When prefixing is in effect (that is, the &@bodyhead1 symbol exists) headings are numbered by DSMHEAD1 instead of by SCRIPT/VS. The prefix, the number and the text of the heading are included in the running footing.

By setting the &@bodyhead1 symbol to null (or to some value) the numbering is done by DSMHEAD1 rather than SCRIPT/VS.

```
.se @bodyhead1 = ''
```

[27] The one exception to this is that the &@xref array is initialized to "1.0" and therefore is not incremented for the very first level one heading in the document. Up to when the first level one heading is encountered the first element of the array indicates a "1." Thereafter it represents the number for the last level one heading.

This line can be put into the profile or into one of the initialization macros. No other modifications are required if this technique is used to capture the heading number in the running footing.

Changing Heading Fonts

The fonts for the headings and for the table of contents entries are all defined in DSMPROF3. There are several sets of font definitions—one for each logical device and the number of the fonts available. Simply changing the font definitions will change the fonts that are used.

Folio by Chapter

For some documents it may be desirable to number the pages in each chapter independently. For example, the pages in the first chapter of the document would be numbered starting with 1-1 and the pages in chapter two would be numbered 2-1, and so on. The first number in the page number would be the level one heading number. This means that level one headings have to be numbered.

Depending on the application that this modification is to be used for, it might be desirable to incorporate this function into a SYSVAR such that users could select page numbering by chapter on the SCRIPT command. Since headings also need to be numbered we could incorporate the page numbering option with the heading number option and use a value of "pnum" for SYSVAR 'H' to indicate both. Then all we need to do is modify the lines in DSM#SETV that process SYSVAR 'H'. These are:

```
.if &E'&SYSVARH eq 0 .se SYSVARH = no
.se *a = index '-NO--YES-NUMBER-' '-&U'&SYSVARH.'
.if &*a ne 0 .se SYSVARH = substr 'no  1.0 1.0' &*a 3
```

We need to change this to:

```
.if &E'&SYSVARH eq 0 .se SYSVARH = no
.se @pnum = no
.if &U'&SYSVARH eq PNUM .se @pnum = yes
.th .se SYSVARH = num
.se *a = index '-NO--YES-NUMBER-' '-&U'&SYSVARH.'
.if &*a ne 0 .se SYSVARH = substr 'no  1.0 1.0' &*a 3
```

The second line initializes the &@pnum symbol to "no." The next line tests if SYSVAR 'H' is "pnum" and if so saves that fact in the &@pnum symbol. Then SYSVAR 'H' is changed to be "num."

The second part of the problem is to change the page numbering. The simplest way to do this is to set the &@bodyhead1 symbol in DSMPROF3 so that it will exist. This will cause the headings to be numbered by the DSMHEAD1 APF. The number can be picked up from the &@xef(1) symbol and used to set the page number.

```
.gs exatt stitle as dsm@shd
.if &@pnum eq yes .pn pref &@xref(1).-
.th .pn 1
```

These lines should be put into the DSMHEAD1 APF.

Putting Level 2 Headings in the Running Footing

The text of the document title and the last head 0 or head 1 are usually reflected in the running footing of a general document. For specific document types you might need to change this to make the running footing reflect the last head 2. If so, the relationship between the level one headings and the level 2 headings needs to be carefully considered. If the current page has been started by a head 1, we won't want to change the running footing to reflect a head 2. Only the first head 2 on a page should affect the running footing, but subsequent head 2's should be remembered and used for the running footing on the next page if there is no intervening head 1.

We are going to have to modify the DSMHEAD1 APF, the DSMHEAD2 APF, and the running footing definition in the profile. We will need to set up several flags to remember the occurrence of a head 1 and a head 2.

The running footing will need to know whether it was caused by a head 1 or not. Therefore we will modify the DSMHEAD1 APF to set a flag just before the .H1 control word and then reset it afterwards.

```
.se @h1@flag 'on
.'h1 ....
.se @h1@flag 'off
```

The DSMHEAD2 APF will need to know whether or not the current page was started by a level one heading. We have the running footing preserve this information in another symbol. We also need the running footing to reset a flag so the DSMHEAD2 APF can tell if it's already changed the footing for the current page. We add the following lines to the running footing definition in the profile:

```
.se @h1start '&@h1@flag
.se @h2flag 'off
```

The DSMHEAD2 APF also needs to be changed. It will always change the value of the &@shead symbol which is the symbol used in the running footing. If the current page wasn't started by a head 1 and this is the first head 2 on the page, we will want to re-execute the running footing to change its contents for the current page. If the page was started by a head 1 or there has already been a head 2 placed on the page, we won't change the running footing for this page. We need to add the following lines to the DSMHEAD2 APF right after the "&@rc2" line: :

```
&@rc2.
.se @shead '&@head
.if &@h2flag eq off .an &@h1start eq off .rf execute
.th .se @h2flag 'on
.se @tg = h
```

First we reset the symbol used in the footing to be the text of the head 2. The next step is to decide if we should re-execute the footing. If the &@h1start symbol is "off" it means that this page wasn't started by a head 1. If the &@h2flag is "off" it means we haven't reset the footing already for this page. Then we change the footing, if the symbol values indicated that it should be changed. Next, we set the &@h2flag to "on" so that that the next time we come into this macro we know that we've already changed the footing for this page. When the footing for the next page is processed, the &@shead symbol contains the value of the last head 2 and the &@h2flag will be set to "off" again and we'll be free to change the footing in the DSMHEAD2 APF.

Formatting Special Characters in Headings

The GML headings tags (and the SCRIPT/VS heading control words) use only a single line of text. This presents problems when you need to include a special character from another font in the text of the heading. For example, suppose you need to include an APL character or a Pi character in your heading. You will need to change fonts, format the special character and restore the previous font. Since the heading must be a single line of text, you can't do this straight forwardly with control words.

However, the .DV [Define Variable] control word provides a simple way to accomplish this. With .DV you can define a variable to be a particular hexadecimal codepoint or string of text in a particular font. The defined variable is then used very much like a symbol is used—with an ampersand in front of the name and a period after the name.

For example, you can define a Pi character as a defined variable, as follows for the 4250 printer:

```
.df pifont type('pi font sans serif' 12) codepage aftc0363
.dv pi font pifont /&x'c6
```

or as follows for the 3800 Printing Subsystem Model 3:

```
.df pifont type('pi sans serif' 12) codepage t1gpi363
.dv pi font pifont /&x'c6
```

Then you can use &pi in the text of your heading.

```
:h1.The Character &pi
```

This same technique can be expanded to simply provide highlighted text within a heading:

```
.dv hd1text font hi1 /word
```

Then you can use &hd1text in the text of your heading to include "word" in highlight level 1.

Defined variables such as these can be used in the text for any GML tag. It is particularly useful for those tags which take only a single line of text—such as definition terms (:DT) and titles (:TITLE).

Paragraphs

Overview

There are four different paragraph APFs in the starter set. The reason for this is that different styles of paragraphs are wanted after headings. DSMPARA is the basic paragraph APF that handles all paragraphs that do not directly follow a heading tag (:H0 - :H6). The :P tag is mapped to the DSMPARA APF in DSMPROF3.

The basic difference between the various paragraph APFs is the presence or absence of skips and indents. The basic assumption is that the first paragraph after a heading should not be indented. After an inline heading (level five and six headings), we want neither a skip nor an indent.

Because no paragraphs are indented in the starter set, you might wonder why we are concerned about indenting or not indenting. The symbol that controls the amount of indention for paragraphs is &@in@p and it is set to 0 in DSMPROF3. It is not difficult to change the amount of indention for paragraphs and we expect that you might want to do so. Therefore, we set up the macros to handle the indention, regardless of the fact that we've got it set to zero.

We could have incorporated all the paragraph variables into one APF that would test various symbols set by the heading macros. The main problem with that approach is that all paragraphs would then have to go through an elaborate series of tests and that would unnecessarily degrade performance. Remapping the :P tag after the headings to a different APF is much faster. The sequence of remapping the :P tag is described below and is illustrated in Figure 15 on page 88.

Paragraph Initialization

DSMPROF3

The profile, DSMPROF3, establishes the basic mapping for the :P tag to the DSMPARA APF. It also defines the amount of indention (&@in@p) for paragraphs and the amount of space to be skipped before the paragraph (&@sk@p). The indention is set to zero and the skip to .75. This is rounded up to 1 line for line devices.

DSMABSTR and DSMPREF

The DSMABSTR and DSMPREF APFs remap the :P tag to the DSMPARA1 APF. This is done because these macros also generate a level one heading and all paragraphs following level one headings are processed with the DSMPARA1 APF. :#np.

DSMHEAD0 and DSMHEAD1

The DSMHEAD0 and the DSMHEAD1 APFs remap the :P tag to the DSMPARA1 APF.

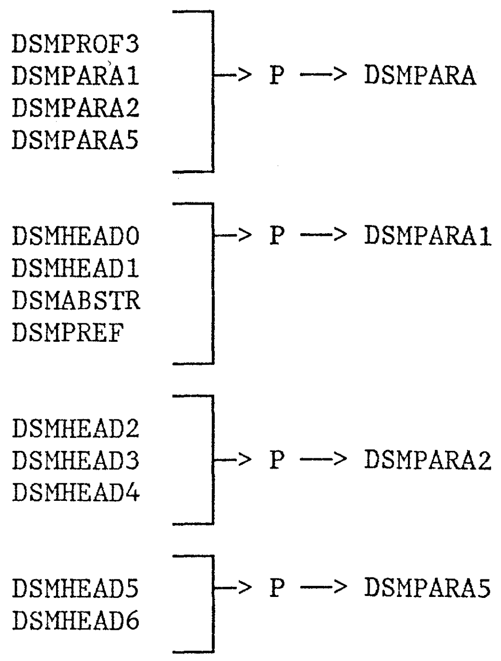


Figure 15. How Paragraph Tags are Mapped

DSMHEAD2, DSMHEAD3 and DSMHEAD4

The DSMHEAD2, DSMHEAD3 and DSMHEAD4 APFs remap the :P tag to the DSMPARA2 APF.

DSMHEAD5 and DSMHEAD6

The DSMHEAD5 and DSMHEAD6 APFs remap the :P tag to the DSMPARA5 APF.

These APFs also define a symbol, &@h5line, that is set to the page number, &$PN, followed by a slash followed by the line number counter, &$LC. This records, roughly, where we are on the page when the level five or six heading is processed. &$PN resolves to the current page number and &$LC resolves to the number of lines left on the page. These two together roughly mark the spot on the page.

The &@h5line symbol is used in the DSMPARA5 APF to determine if there was any text between the heading and the next paragraph tag. If any text is processed in between the :H5 or :H6 tag and the next paragraph, one of these symbol values will probably have changed. We need to know this because we need to know whether to generate the colon with the :P tag. If the :P tag isn't directly after the heading, we don't want a colon.

Paragraph Processing

DSMPARA

Most paragraphs in a document are processed by the DSMPARA APF. The :P tag is mapped to DSMPARA initially by DSMPROF3. The DSMPARA APF performs the following processing:

1. Skips according to &@sk@p (set in DSMPROF3 to 1).
2. Indents the first line of the paragraph according to +&@in@p (set in DSMPROF3 to 0). This is an incremental indention over and above any indention currently in process. It is incremental because we want paragraphs to work in list items, long quotes, and all elements that cause indention.

DSMPARA1 and DSMPARA2

In the starter set, there is no difference at all between the DSMPARA1 APF and the DSMPARA2 APF. There are two different APFs in order to facilitate making a difference between paragraphs following the different types of headings. As we'll see in "Modifications to Paragraphs" on page 90 below, this makes it easy to change the handling of the paragraph following the level zero and one headings without disturbing paragraphs after other headings.

The first paragraph after a level zero or one heading (:H0 or :H1 tag or document section tag, such as :ABSTRACT or :PREFACE) is processed by the DSMPARA1 APF. The :P tag is remapped to DSMPARA1 when a :H0 or :H1 heading is processed.

The DSMPARA2 APF is used to process the first paragraph after level two through four headings (:H2, :H3 and :H4 tags). The :P tag is remapped to DSMPARA2 when a :H2-4 heading is processed.

The DSMPARA1 and DSMPARA2 APFs perform the following processing:

1. Skips &@sk@p (set to 1 by DSMPROF3).
2. Does not indent (making it different from DSMPARA)
3. Remaps the :P tag to DSMPARA because after the first paragraph following a heading, we'll want normal style paragraphs.

DSMPARA5

Paragraphs that follow level five and six headings (:H5 and :H6 tags) are processed by the DSMPARA5 APF. The primary purpose of this APF is to append a colon to the heading and process the paragraph without any skips or indents.

The DSMPARA5 APF performs the following functions:

1. The colon that follows the heading needs to be formatted in the same font as the heading. Since both level six and level five headings are following by paragraphs produced by the DSMPARA5 APF, we needed some way of telling which font to start—the "hd5" font or the "hd6" font. The DSMHEAD5 and DSMHEAD6 APFs save the appropriate font number in a symbol named &@para5@fnt, which when combined with "hd" yields the proper .BF [Begin Font] control word:

```
.bf hd&@para5@fnt
.bf hd5                 or              .bf hd6
```

2. The page number (&$PN), followed by a slash, followed by the line count (&$LC) is compared to the &@h5line symbol that was saved in the DSMHEAD5 or DSMHEAD6 APF.

If text has been processed between the heading and the paragraph tag, the values of these two symbols (&$PN and &$LC) will probably not be the same as they were when they were saved in the &@h5line symbol. This doesn't always work, because the line count value (&$LC) is not always incremented directly after processing the text. However, we need some way to tell if the :P tag directly followed the heading, and this is the best approximation available.

3. If there has been no intervening text:

   a. The correct font is started.

   b. A colon is appended to the text of the heading, using the .CT [Continued Text] control word.

   c. The previous font is restored.

4. If there has been intervening text, DSMPARA is called to finish processing the paragraph. If text or other tags have been put between the heading and the paragraph tag, this indicates a regular style paragraph—it no longer follows a heading. It is processed with the normal paragraph APF, DSMPARA.

5. In either case, the :P tag is remapped to the DSMPARA APF.

DSMPCONT

The DSMPCONT APF processes the :PC tag. A paragraph continuation occurs when a paragraph is interrupted by an example (:XMP) or a figure (:FIG) and it is formatted by doing a skip (&@sk@p) but not an indent.

Modifications to Paragraphs

Changing Indention and Spacing

The amount of indention for the first line of a paragraph and the amount of skip before it are controlled by two symbols that are set in DSMPROF3—&@in@p controls the indention and &@sk@p controls the amount of space skipped. The original settings are 0 and .75 respectively. To change these values all you need to do is change their settings in DSMPROF3.

Using Large Initial Capitals

Creating large display initials at the beginning of a paragraph involves using a large font for the first letter of a paragraph. This is possible only on page printers, of course. First we need to decide when to use it because we probably wouldn't want every paragraph to have a display initial.

Let's suppose that we are going to make only the paragraphs that directly follow a level zero or one heading have large initial capitals. That means that all we have to do is alter the DSMPARA1 APF to do the job and we're done. Suppose, furthermore that we are going to let the user control whether he wants large initial caps or not by using a new SYSVAR on the SCRIPT command.

Isolating the First Letter: First, let's modify the DSMPARA1 APF to create the initial capital letter. If we are not formatting for a page printer (SYSOUT is not PAGE), then we can end the macro because there is no way to produce a large initial cap on a line printer. If we can create the large initial caps, we start with a keep that should encompass the large initial capital, plus some text. Next, we get the residual text into a local symbol so that we can pick the first character out of it. Then we turn off symbol substitution so that we can use the SUBSTR function of .SE [Set Symbol] without worrying about whether there are blanks in the residual text line. Then we split the line into two separate symbols. The first symbol will contain the uppercase letter we are going to put in the large font and the other symbol will contain the rest of the line.

```
.sk &@sk@p
.aa p dsmpara
.if SYSOUT ne PAGE .me
.kp 1.2i
.gs scan *line
.su off
.se *cap = substr &*line. 1 1
.se *cap = &U'&*cap
.se *rest = substr &*line. 2
```

Notice that we didn't specify the length parameter on the second substring function. This parameter defaults to the rest of the line if it isn't specified, and that's what we want.

Calculating the Baseline Shift: The next step is to figure out how much we need to drop the baseline to get the top of the large capital letter to line up with the rest of the line. Let's look at some dimensions that we have available and figure out how much we need to shift down.

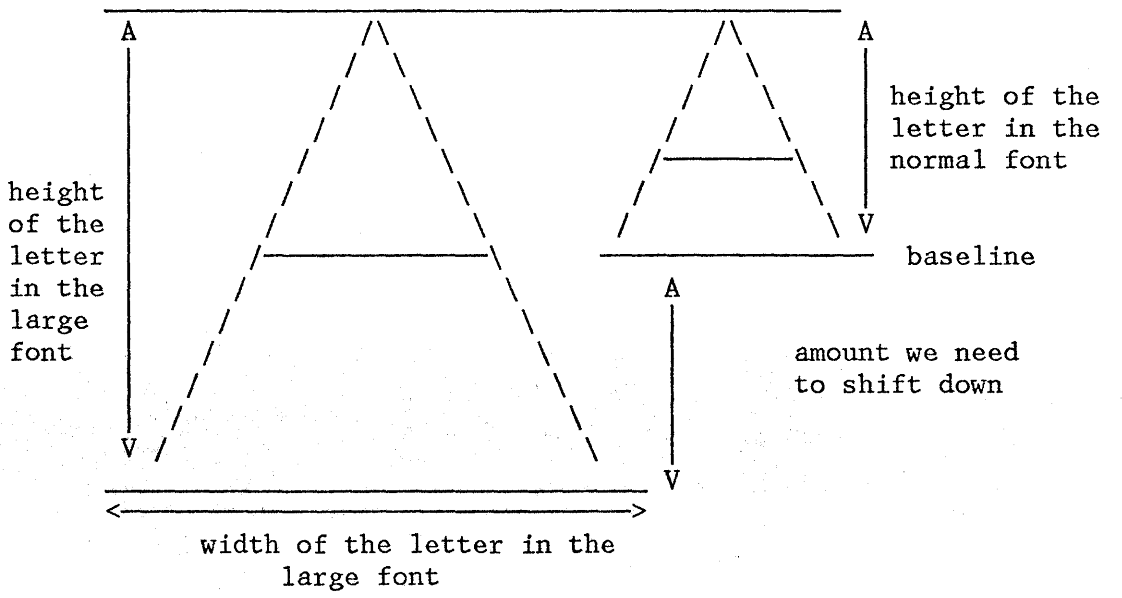

height of the letter in the large font
height of the letter in the normal font
baseline
amount we need to shift down
width of the letter in the large font

Figure 16. Formatting Large Initial Capitals for Paragraphs

From Figure 16 you can see that the amount we want to shift down is the difference between the height of the capital and the height of the rest of the letters on the line. To get the height of a letter we can use the &DV symbol attribute and the "mv" space unit notation. We can't simply use the line spacing of the font because we don't want to take into account any white space on the top of the letters. The em-height (mv) space notation returns the actual height of the characters in the current font.

The font we are using at the moment we do the calculation is important. We will need to use a new font to format the capital and to calculate its height, but we must first calculate the height of the rest of the line before we change fonts.

```
.se *basehgt = &DV'1mv
.bf large
.se *shift = &DV'1mv - &*basehgt
.su on
.sb -&*shift.dv
```

We've calculated the height of the letter in the normal font (&*basehgt), started the "large" font and calculated the difference between the current height of the letter (in the large font) and the height of the letter in the normal font. Then we use this value (&*shift) to shift the baseline down using the .SB [Shift Baseline] control word.

Calculating the Indention: Now we need to figure out how much to indent the lines after the first one. From Figure 16 on page 91 we can see that the amount the second line is indented is the width of the capital letter. We can set up a delayed indent using the &W' symbol attribute and the AFTER parameter on the .IN [Indent] control word to avoid indenting the first line. We can avoid indenting the first line by delaying the indent for a little—one device unit (1dv) will do just fine.

We don't want to indent for the rest of the paragraph, so we need to know how many lines will need to be indented. We don't know that, but we do know how much vertical space they take—the same amount that we shifted down and that's how long the indention should last.

Then we can set the indent:

```
.in +&DH'&W'&*cap..dh for &*shift.dv after 1dv
```

Now we need to format the capital.

```
&*cap
```

Then we:

1. Reset the baseline to what it was
2. Return to the original font
3. Complete the rest of the line, forcing continuation with the initial capital.

```
.sb +&*shift.dv
.pf
.ct &*rest
```

The New DSMPARA1 Macro: Here's what the DSMPARA1 APF looks like when we are through:

```
.sk &@sk@p
.aa p dsmpara
.if SYSOUT ne PAGE .me
.kp 1.2i
.gs scan *line
.su off
.se *cap = substr &*line. 1 1
.se *cap = &U'&*cap
.se *rest = substr &*line. 2
.se *basehgt = &DV'1mv
.bf large
.se *shift = &DV'1mv - &*basehgt
.su on
.sb -&*shift.dv
.in &DH'&W'&*cap..dh for &*shift.dv after 1dv
&*cap
.sb +&*shift.dv
.pf
.ct &*rest
```

Now, how do we handle the selectable portion of the problem? We want to do this fancy stuff only when the user requests it with a SYSVAR. First, let's make up a SYSVAR—"I"—for initial.

The lines we just added to the macro only apply if &SYSVARI is "yes." If &SYSVARI isn't "yes," we just exit the APF.

```
.sk &@sk@p
.aa p dsmpara
.if &SYSVARI ne yes .me
```

That's all we need to do to our new DSMPARA1 APF. Two more things remain to be done. First, we must define the font named "large" that we have used in the new DSMPARA1 APF. The font definition,

```
.df large type(24)
```

defined a 24 point font named "large." Put this .DF [Define Font] control word in the profile.

The last step is to process the &SYSVARI value given on the command to make sure that "YES" and "Yes" and "yes" all end up with &SYSVARI set to "yes," because that is what we are testing for. We also need to establish the default value of &SYSVARI as "no." This processing needs to be done in the DSM#SETV macro:

```
.se *a =        index '-YES-NO-' '-&U'&SYSVARI.'
.if &*a eq 0 .se *a = 5
.se SYSVARI = substr 'yes no ' &*a 3
```

See "Validating Keywords" on page 13 for a detailed explanation of the technique used here.

Creating Numbered Paragraphs

Paragraphs are not numbered in a general document. However, suppose we need to create a document that contains numbered paragraphs that look like this:

```
1.01  Notice  that the text  of the
      paragraph is  indented for  a
single line following the paragraph
number.
```

In order to create something like this, we have to make a few assumptions:

1. The paragraph numbers never go above 99.
2. The first digit of the number is derived from a level one heading.
3. There are also normal, non-numbered paragraphs in the document.

The last assumption means that we need to create an entirely new tag, :PNUM, to mark up the numbered paragraphs, leaving the :P tag for the normal paragraphs.

Initialization: Since the numbered paragraphs are tied to the headings, the first thing we need to do is define that relationship and modify the heading definitions (.DH [Define Head Level]) accordingly along with the DSMHEAD1 APF. First, we redefine the headings (probably in the profile) to never be numbered. We're going to number the level one headings ourselves in the DSMHEAD1 APF and it would be undesirable to have the lower level headings numbered if the paragraphs were numbered. We might also want to change the level one heading to not cause a section break or a page eject and to have a skip before it. These kinds of decisions would be a function of exactly how the headings were to look. By changing the level one heading in this way, it becomes similar to a level two heading:

```
.dh 1 npa nosect nonum skbf 2
.dh 2 nonum
.dh 3 nonum
.dh 4 nonum
```

We also need to initialize a paragraph number symbol and a heading number symbol in the profile:

```
.se pct = 0
.se @hdnum = 0
```

Modifying the DSMHEAD1 APF: The next step is to do two things to the DSMHEAD1 APF:

1. Number the headings
2. Reset the paragraph number symbol to zero.

The new DSMHEAD1 APF might look like this:

```
.dsm#rset H.-1
.dsm#dup1
.gs scan @head
.se @hdnum = &@hdnum + 1
.'se @head '&@head1 &@hdnum.  &@head
.'se @shead '&@head
.gs exatt stitle as DSM@SHD
.'h1 &@head
.se @tg = h
.gs exatt id as DSM@IDS
.se pct = 0
.aa p dsmpara1
```

We added several .SE [Set Symbol] control word lines to manipulate the heading number and the paragraph number and to include the heading number in the text of the heading. We've also deleted two lines from the DSMHEAD1 APF which were used to increment the heading number with the .GS [GML Services] HCTR control word. These lines are no longer needed because we are going to handle the number ourselves.

Creating the New APF: The next step is to create a new APF to process the :PNUM tag. We'll name it PNUM to simplify matters.[28] Its function is to calculate the paragraph number and establish some indention to make the number stand out from the text of the paragraph. It will also skip a space before the paragraph as the normal paragraph APF does.

```
.sk &@sk@p
.se pct = &pct + 1
.if &pct lt 10 .se pct '0&pct
.se pnum '&@hdnum..&pct
.in 6 after 1 for 1
&pnum.&$RB.&$RB.&$CONT.
```

If we are formatting for a page printer, we need to be a little fancier about calculating the indention. Instead of setting an indention of 6, we need to calculate the width of the number plus the required blanks.

```
.sk &@sk@p
.se pct = &pct + 1
.if &pct lt 10 .se pct '0&pct
.se pnum '&@hdnum..&pct
.se *w = &DH'&W'&$RB * 2 + &DH'&W'&pnum
.in &*w.dh for 1 after 1
&pnum.&$RB.&$RB.&$CONT.
```

We could even get fancier and use a slightly larger font for the paragraph number for a page printer. First we need to define a large font to use for the paragraph numbers. It could be something like this:

```
.df parafont type(18)
```

This line should be put in the profile along with all the other font definitions.

[28] By naming the APF the same as the tag, we avoid having to include a .AA [Associate APF] control word to associate the tag with the appropriate APF.

Then we need to perform the calculations explained in the discussion about using large initial capitals on a paragraph. This involves calculating the difference between the heights of the normal text font and the large initial cap font. This difference represents the amount we need to shift the baseline down in order to align the tops of the characters.

```
    .sk &@sk@p
    .se pct = &pct + 1
    .if &pct lt 10 .se pct '0&pct
    .se pnum '&@hdnum..&pct
    .se *basehgt = &DV'1mv
    .bf parafont
    .se *shift = &DV'1mv - &*basehgt
    .sb -&*shift.dv
    .se *w = &DH'&W'&$RB * 2 + &DH'&W'&pnum
    &pnum.&$RB.&$RB.&$CONT
    .sb +&*shift.dv
    .pf
    .in &*w.dh after 1 nobreak for &*shift.dv
```

In this case we shifted down the difference in line spacing between the two fonts we are using. This more or less lines up the tops of the characters.

We have also changed the indent line and moved it. We have to be careful about what font we are using when we use the "after 1" parameter. The size of the "1" will be determined by the current font. Therefore, we want to wait and not set the indention until after we have returned to the body font with the .PF [Previous Font] control word. That is why we moved the indent control word further down in the macro.

However, starting an indent when we have a partial output line (the partial line contains the paragraph number and two required blanks) causes a line break to occur. To avoid breaking the line after the number, we added the NOBREAK option to .IN [Indent].

There's one more problem with our indention: If the value of &*shift is zero (which will always be the case for line printers), the indention becomes permanent rather than temporary. In other words, FOR 0 is just the same as not using the FOR parameter at all. To solve this problem we need to check that &*shift is at least equal to one.

```
    .if &*shift le 1 .se *shift = 1
    .in &*w.dh after 1 nobreak for &*shift.dv
```

A Word of Caution: Numbered paragraphs as implemented here are fairly simple. They are also of limited use. To really implement them and use them within the starter set, you need to decide which heading level the numbers should function off of, which headings will be allowed in the document and how they should look. The implementation outlined above has been greatly simplified. Lower level headings have not been adjusted except to inhibit numbering. Other adjustments might be necessary depending on the nature of the application.

List Processing

There are many different macros which are called upon to process the various parts of a "list" depending upon the type of list. The diagram below depicts the basic relationship between the primary macros used. The DSM#LTYP and DSM#LINT macros, called from DSMLISTM, initialize the formatting environment for the list and the various parts of the list can be processed by special APFs such as DSMLPART, DSMLITEM, DSMDDEF and so on. The DSMELIST APF restores the normal formatting environment.

```
*----------*
| DSMSLIST |
| DSMGLIST |   *----------*    *----------*    *----------*
| DSMOLIST |-->| DSMLISTM |--->| DSM#LTYP |--->| DSM#LINT |
| DSMDLIST |   *----------*    *----------*    *----------*
| DSMULIST |        |
*----------*        |
                    |--> Defines:  @TSIZE macro
                    |              @TERMHI macro
                    |              @HEADHI macro
                    |
                    *--> Maps:     :LI ---------->  DSMLITEM

                                   :DDHD -------->  DSMDDHD
                                   :DTHD            DSMDTHD
                                   :DT ---------->  DSMDTERM
                                   :DD ---------->  DSMDDEF

                                   :GT ---------->  DSMGTERM
                                   :GD ---------->  DSMGDEF

                                   :LP ---------->  DSMLPART
```

Figure 17. Primary List Macros: This diagram shows the calling sequence among the primary list macros and APFs.

List Initialization

DSMPROF3

The profile performs a great deal of initialization for lists, including:

1. The indention and space values pertaining to each type of list are defined (for example, &@in@o and &@sk@o for ordered lists). See Figure 18 on page 99 for the list of control symbols.
2. The default highlight levels for definition headings (&@hi@h) and definition terms (&@hi@d) are set to 3 and 2, respectively. The default highlight level for glossary list terms (&@hi@g) is 2.
3. The &@olistnest and &@ulistnest symbols are defined. These symbols control the sequence of list item identifiers that are to be used for each level of list nesting.
4. The list item identifiers are defined in DSMPROF3. See "Starter Set Initialization" on page 19 for about how this works.
5. The list tags (:DL, :GL, :OL, :SL and :UL) are mapped to their appropriate APFs.

DSM#SET

This macro also performs some important initialization for lists, including:

1. The list nesting symbol, &@nest@l, is set to zero. This symbol is used to keep track of how many levels of lists are open.
2. The &@sk@l symbol is set to &@sk@ls which is set in DSMPROF3 to .75. This symbol governs the amount of space that is skipped before and after lists.
3. The &@nest@o and &@nest@u symbols are undefined. These two symbols keep track of the current level of unordered and ordered lists.

Getting the List Started

There are five types of lists supported by the starter set. Each type of list has its own tag and is mapped to its own APF. These are:

```
definition   (DSMDLIST)   (:DL)
glossary     (DSMGLIST)   (:GL)
ordered      (DSMOLIST)   (:OL)
simple       (DSMSLIST)   (:SL)
unordered    (DSMULIST)   (:UL)
```

However, all five of the APFs for these tags do the same thing: immediately call the DSMLISTM macro. The first parameter on the call to DSMLISTM is a single character which denotes what kind of list this it. The possibilities are "*" for definition, "g" for glossary, "o" for ordered, "s" for simple and "u" for unordered, The APFs also pass along to DSMLISTM the parameters (value attributes) that were passed to them (&*).

Nesting Controls Symbols

@olistnest	id sequence (OL)
@ulistnest	id sequence (UL)
@denest@o	denest sequence (OL)
@denest@u	denest sequence (UL)
@nest@o	nest level (OL)
@nest@u	nest level (UL)
@renest@o	renest sequence (OL)
@renest@u	renest sequence (UL)

Font Control Symbols

@hi@l	term highlight
@hi@hd	heading highlight
@hi@h	heading default
@hi@g	glossary default
@hi@d	definition term default
@nest@l	saves current list control symbols

Miscellaneous List Control Symbols

@item#	list item number
@ltype	current list type (o,u,s,d or g)

Formatting Control Symbols

@in	indention before list
@in@l	indent for current list
@in@o	default ordered indent
@in@u	default unordered indent
@in@s	default simple indent
@in@g	default glossary indent
@in@d	default definition
@li@tab	tab to start list item
@sk@l	skip for current list
@sk@o	default ordered skip
@sk@u	default unordered skip
@sk@s	default simple skip
@sk@d	default definition skip
@sk@g	default glossary skip
@break	break attribute value

Figure 18. List Symbols: Many different symbols are used to control list nesting and list formatting. Some are initialized in the profile. Some are initialized in the DSMLISTM, DSM#LTYP, and the DSM#LINT macros. They are used throughout the list macros to determine what the situation is and to control the formatting.

DSMLISTM

The DSMLISTM macro sets up the formatting environment for list processing. It is called by the DSMDLIST, DSMGLIST, DSMOLIST, DSMSLIST and DSMULIST APFs which process the :DL, :GL, :OL, :SL and :UL tags, respectively. It is also the APF for the :L tag which is mapped directly to DSMLISTM in DSMPROF3.[29]

The DSMLISTM macro performs the following processing:

1. Defines macros to process the TERMHI, TSIZE and HEADHI attributes. These save the attribute value in the &@hi@l, &@in@l and &@hi@hd symbols, respectively.

   The lines that define the macros are simultaneously removed from DSMLISTM. See "Special Techniques" on page 13 for details on self-modifying macros.

2. Skips a line (&@sk@l) conditionally.

3. Saves the values of several variables in an element of the &@nest@l symbol array. This is done each time a list is started. This is particularly critical when there is already a list going as these symbol values need to be restored when the outer list is restarted. The symbol values saved include:

[29] The :L tag is not formally part of the starter set, which is why it is not documented. However, it is used to create lists which use an asterisk as the list item identifier.

- The list type (&@ltype),
- The current item number (&@item#),
- The indention values (&@in and &@in@l),
- The skip value (&@sk@l),
- The term highlight level (&@hi@l),
- The heading highlight level (&@hi@hd), and
- The current value of the BREAK attribute (&@break).

The indention for lists is done in two parts as is illustrated in Figure 19. The first indent is to where the identifier or number goes and the second indent (which is incremental) is to where the text starts.

4. Calculates the indention value. If this is not the first level of list, the &@in value contains the base indention for the previous list level and the &@in@l symbol will contain the incremental amount for the previous list. If these symbols exist, we indent their value. The total amount of indention, now in &$IN, is saved in &@in which now represents the base indention for the current list starts.

5. Calls DSM#LTYP to get the &@ltype and &@id@l symbols set up.

6. Processes the parameters passed in (&*) to see if either COMPACT or BREAK was specified. If COMPACT is found, &@sk@l is set to 0 to cause no extra spaces between list items and &*c is set to 0. If BREAK is found, the &@break symbol is set to BREAK. This symbol will be used on the .IS [Inline Space] TO control word line in the definition list macro. In other words the BREAK attribute function is nothing more than the BREAK parameter on the .IS [Inline Space] control word.

```
| 1.     | This is a first level list item.  The identifier
|        | for the item is '1.'.  The text of the item
|<----->| starts at an indention of &@in@l.
|&@in@l  |
|        |
|        |
|        | a.      | This is a second level list item.  The
|<----->|<------>| indention for its text is the sum of the
| &@in   | &@in@l  | indention for the first level list (&@in)
|        |         | and the incremental amount (&@in@l).
|        |         |
|        |         |
|                  | 1)     | This is a third level list item.
|<--------------->|<----->| It's indention is the sum of the
|      &@in        |&@in@l  | first and second level indention
|                  |        | (&@in) plus an incremental
|                  |        | indention (&@in@l).
```

Figure 19. List Indention: The indention for each level of list is calculated in two parts. The first indention, &@in, represents where the list item identifier goes. It is the sum of the indention for the previous levels of list. The second indention, &@in@l, is the incremental indention necessary to get from where the identifier is placed to where the text of the item should start.

A detailed explanation of the technique used here to determine if the attributes were specified or not can be found in "Special Techniques" on page 13.

7. Sets up the formatting environment for this list level.

   a. Sets the item counter, &@item#, to 0. This will be incremented each time a :LI tag is processed.

   b. Sets the highlight font for terms, &@hi@l, based on list type.

   c. Sets &@in@l based on list type. This will be the incremental indention from where the identifier or term is placed to where the list item text or description starts.

   d. Sets the highlight font for headings (&@hi@hd) using the &@hi@h symbol that was defined in DSMPROF3.

   e. Sets &@sk@l based on list type and whether COMPACT was found. If COMPACT was not specified, the &*c symbol will be null causing the &@sk@&@ltype value to be used. If COMPACT was specified, the &*c value (0) will be used.

8. Processes the TSIZE attribute using the @TSIZE macro defined above. The TERMHI attribute is processed using the @TERMHI macro defined above and the HEADHI attribute is processed using the @HEADHI macro defined above.[30] These attributes will override the &@hi@l, &@hi@hd or &@in@l symbols whose defaults were just set above.

9. Performs the base indention and incremental indention (&@in and &@in@l). The total indention value is saved in &@li@tab. This value represents where the text of the list item or the definition description starts. It is used on a .IS [Inline Space] TO control word by the list item and definition description APFs.

10. Maps the :LP tag for list parts to its APF, DSMLPART.

11. For glossary lists, maps the :GD and :GT tags to their respective APFs, DSMGDEF and DSMGTERM. For definition lists the :DTHD, :DDHD, :DT and :DD tags are mapped to DSMDTHD, DSMDDHD, DSMDTERM, and DSMDDEF.

12. For all other types of lists, maps the :LI tag to the DSMLITEM APF.

DSM#LTYP

The DSM#LTYP macro decides what type of list is being processed. It is called by the DSMLISTM macro with a single character as a parameter in &*1. The character will be either g, o, s, u, or *. The asterisk is used to denote a definition list.

DSM#LTYP performs the following functions:

1. Calls the DSM#LINT macro which defines the &@denest@u, &@denest@o, &@renest@u and &@renest@o symbols. These symbols contain strings of numbers which will control the way in which nested levels of lists are handled.

2. Determines from the single character[31] (g,o,s,u,*) that is passed to the macro what type of list is being processed.

   a. Picks out the first character of the first parameter.
   b. Checks if it's an O, U, S, G or *.
   c. If none of the above, assumes it's "undefined" (z).
   d. Sets &@ltype to d, g, o, s, u or z (for undefined).

[30] "thi" is recognized as a synonym for the TERMHI attribute and "hhi" is recognized as a synonym for the HEADHI attribute.

[31] The asterisk (*) denotes definition lists.

3. For g lists, d lists, and s lists, sets the &@id@l symbol to null. This symbol is supposed to contain the necessary symbol function to produce the list item identifier. For glossary lists and definition lists the identifier portion (left part) of the list is a term and for simple lists there is no identifier.

4. For unordered lists DSM#LTYP:

   a. Makes sure that the &@nest@u symbol exists. This symbol indicates the level of nesting of unordered lists. If there are no unordered lists open already, the symbol won't exist and we set it to the length of &@ulistnest. This is the number of different identifiers defined for each level of nested lists. We will use this value to select the correct identifier to use for the list we are starting.

   b. Resets &@nest@u to one of the values in the &@renest@u symbol. This symbol was defined in DSM#LINT to be "234561." For the first level of list, the &@nest@u symbol will be 6, so we will select the sixth number, "1." For the second level of unordered list we will select the first number, "2" because the value of &@nest@u will be "1" indicating that there is already one level of list going.

   c. Determines the identifier number. The &@ulistnest symbol contains the proper sequence in which we should use the item identifiers that have been defined. It will be "123" for line printers and "12345" for page printers unless the profile has been modified. Using the &@nest@u symbol value calculated above, we select the proper identifier value out of &@ulistnest. So for the first unordered list, we'd pick the first number, and use it to construct the identifier number.

5. Performs the same processing as described above for ordered lists except that the &@renest@o and &@olistnest symbols are used.

6. Determines the appropriate item identifier using the number just calculated above, for both unordered and ordered lists. The identifiers are in defined variables named &@id@l@u1, &@id@l@u2, and so on. The number just calculated is the last part of the identifier symbol name. The list type determined above is the next to last part of the symbol name (&@ltype).

   For example for the third level unordered list the identifier symbol, &@id@l, is defined as follows:

   ```
   .se *a = &@ltype.&*a
   .se *a = &@ltype.3
   .se *a = u3

   .se @id@l '&V'&@id@l@&@ltype.&*a..'
   .se @id@l '&V'&@id@l@u&*a..'
   .se @id@l '&V'&@id@l@u3.'
   ```

7. Issues a message if the identifier symbol isn't defined. This indicates that the &@olistnest or &@ulistnest symbols are out of sync with the identifier definitions in DSMPROF3.

8. Puts the value of the identifier symbol into &@id@l.

DSM#LINT

The DSM#LINT macro defines the symbols which control the nesting and de-nesting of the various kinds of lists. It is called by DSM#LTYP.

There are two denesting symbols and two renesting symbols—one set for ordered lists and one set for unordered list. Their values are based on the setting of &@ulistnest and &@olistnest which are defined in the profile.

Suppose that the &@olistnest symbol is set to "123456" meaning that there are six list item identifiers defined which are numbered 1 through 6. These identifiers are to be used in the order given in &@olistnest. You can change the sequence of the identifiers by simply reordered the numbers in &@olistnest. The &@denest@o symbol will be defined as follows:

```
.se @denest@o = substr &L'&@olistnest.12345678 1 &L'&@olistnest
.se @denest@o = substr 612345678 1 6
.se @denest@o = 612345
```

The &@denest@o symbol will be used in the DSMELIST APF to figure out the proper level of the previous ordered list when we are ending an ordered list.

Using the current level of nesting as a position in the &@denest@o symbol we can derive the level of the previous list. For example, when the third level of list is being ended we can tell that the second level needs to be restarted. That seems very straight forward, however, the levels actually wrap around such that the seventh level of list is the same as the first level in terms of which identifiers are used. In this case the value we will pick out of &@denest@o will be "6" which is what we want. See the discussion of the DSMELIST APF below for more details on how this all works.

The &@renest@o symbol is used to figure out the proper level of the next level of ordered list when we are starting one in the DSMLISTM macro. It is used in the DSM#LTYP macro to select the proper list item identifier for an ordered list when it is started. It is set as follows in the DSM#LINT macro:

```
.se @renest@o = substr &@denest@o.&L'&@olistnest.1 3 &L'&@olistnest
.se @renest@o = substr 61234561 3 6
.se @renest@o = 234561
```

Using the current level of nesting as a position in the &@renest@o symbol we can derive the level of the next list. For example, when the current list is the sixth level, we can tell that the next level will be like the first level in terms of which identifiers are used. In this case the value we will pick out of &@renest@o will be "1" which is what we want. See the discussion of the DSM#LTYP macro above for more details on how this symbol is used.

Processing Items on the List

DSMLITEM

The DSMLITEM APF processes the :LI tag. It skips, indents and prints the item identifier (if any), as follows:

1. Increments the list item counter (&@item#).

2. Skips and indents (&@sk@l and &@in@l—set in DSMLISTM). &@in is the base indention for the current level of list. &@in@l is the incremental amount of indention we need for the current list.

   The indent is done in two parts—indent and then an incremental indent after 1. The first indent controls the placement of the identifier and the second controls the placement of subsequent lines of text. The initial line of text is placed with an immediate tab (.IS [Inline Space] TO n). If this is a second level list, &@in will be 4 due to the indention of the first level and &@in@l will also be 4 representing the amount of indention for this level.

3. Sets the &@tg symbol to "d" to indicate that an ID for a list item is being processed. The DSM@IDS macro is then called to process the ID attribute. This macro is described in "Cross-References" on page 147.

4. Puts out the identifier that is in &@id@l. This symbol was also set up in the DSM#LTYP macro when the list was started.

5. Moves over to where we want to start the text of the list item. The .IS [Inline Space] TO control word is used to avoid using the tab rack. The position to move to is in &@li@tab which was defined in DSMLISTM when the list was started. It will get us to the same place we defined with the delayed indention.

DSMDTHD

The :DTHD tag is processed by the DSMDTHD APF. The definition list headings are actually produced by the DSMDDHD APF. The DSMDTHD APF saves the residual text of the tag in &@dthead for processing later by the DSMDDHD APF. It also skips a line.

DSMDDHD

The :DDHD tag is processed as follows by the DSMDDHD APF:

1. Determines whether or not a :DTHD tag was processed. If so, the &@dthead symbol will exist and will contain the heading for the terms. If it doesn't exist, we set it to null and there won't be any heading for the terms.

2. Starts a keep around the headings that will ensure that the headings are kept with the first definition term. The size of the keep depends on whether or not we are producing a compact list. The &@sk@l symbol will be zero if the list is compact. Otherwise it will be one. We calculate the depth of the keep of 3 lines plus the depth of the skip.

3. Skips and indents (&@sk@l and &@in@l—set in DSMLISTM). &@in is the base indention for the current level of list. &@in@l is the incremental amount of indention we need for the current list.

   The indent is done in two parts—indent and then an incremental indent after 1. The first indent controls the placement of the term and the second controls the placement of the description headings. The initial line of text is placed with an immediate tab (.IS [Inline Space] TO n). If this is a second level definition list, &@in will be 10 due to the indention of the first level and &@in@l will also be 10 representing the amount of indention for this level.

4. Obtains the residual text in the &*ddhead local symbol.

5. Starts the highlight font for definition list headings. The &@hi@hd symbol contains the number of the highlight font to use. This is either the default (set to 3 in DSMPROF3) or it has been set by the user with the HEADHI attribute.

6. Puts out the term heading using literal mode just in case it should start with a period.

7. Performs a .IS [Inline Space] TO control word to get to where the description heading should start. The &@li@tab symbol was calculated in DSMLISTM and is the sum of the two indentions described above.

8. Puts out the description heading using literal mode in case it happens to start with a period.

9. Restores the previous font.

10. Skips another line to separate the headings from the first term.

11. Undefines the &@dthead symbol because we don't need it any more.

DSMDTERM

The :DT tag is processed by the DSMDTERM APF which saves the definition term for processing later by the DSMDDEF APF. DSMDTERM performs the following processing:

1. Checks that there are no unprocessed definition terms left over from before. The term will be put in the &@id@l symbol here. When it is placed on the page, the &@id@l symbol is undefined. Therefore, if the &@id@l symbol exists it means that we've gotten two definition terms without an intervening description. In this case, we issue an error message.

2. Checks if there is an unprocessed term heading. The &@dthead symbol is used for the term heading and is undefined when the heading is placed on the page by the DSMDDHD APF. If this symbol exists it means that a term heading has been processed but there was no description heading. In that case, rather than assume that that is an error we will put out the term heading.

   a. Calculate a keep of 3 lines plus the depth of &@sk@l.
   b. Starts a keep.
   c. Skips a line conditionally.
   d. Indents to where the heading should start.
   e. Starts the definition heading font (or the current font).
   f. Starts literal mode for one line.
   g. Formats the term heading.
   h. Restores the previous font.
   i. Skips a line.
   j. Undefines the &@dthead symbol.

3. Skips a line conditionally before the term.

4. Scans to get the residual text of the :DT tag into the &@id@l symbol. The term will be placed on the page by the DSMDDEF APF.

DSMDDEF

The :DD tag is processed by the DSMDDEF APF which formats the definition term and sets up the processing environment for the definition description with the following processing:

1. Checks for a missing definition term, if so, inserts "?" for the term and calls DSM#MSG to issue a message that a definition list term is missing.

2. Checks if there is an unprocessed term heading. The &@dthead symbol is used for the term heading and is undefined when the heading is placed on the page by the DSMDDHD APF. If this symbol still exists it means that a term heading has been processed but there was no description heading and no definition term. In that case, rather than assume that that is an error we will put out the term heading.

   a. Calculates a keep of 3 lines plus the depth of &@sk@l.
   b. Starts a keep.
   c. Skips a line conditionally.
   d. Indents to where the heading should start.
   e. Starts the definition heading font (or the current font).
   f. Starts literal mode for one line.
   g. Formats the term heading.
   h. Restores the previous font.
   i. Skips a line conditionally.
   j. Undefines the &@dthead symbol.

3. Skips conditionally (&@sk@l).

4. Indents the base indention (&@in) and the incremental delayed indention (&@in@l).

5. Starts the highlight font for definition terms. The &@hi@l symbol contains the number of the highlight font to use. This is either the default (set to 2 in DSMPROF3) or it has been set by the user with the TERMHI attribute

6. Formats the definition term which was put into the &@id@l symbol by the :DT tag.

7. Restores the previous font.

8. Inserts space to where the definition description should start. The &@li@tab symbol was calculated in the DSMLISTM macro when the list was started. The &@break symbol may be set to "break" or to null depending on whether or not the BREAK attribute was specified on the :DL tag.

9. Sets &@id@l to null.

DSMGTERM

The :GT tag is processed by the DSMGTERM APF which saves the glossary term specified with the :GT tag, as follows:

1. Checks if there is already a glossary term defined which has not been fully processed and calls DSM#MSG to issue a message if there is. This would happen only if there were two :GT tags in a row with no intervening :GD tag.

2. Skips a line. The value of &@sk@l may be zero if this is a compact glossary list.

3. Scans to get the term into the &@id@l symbol.

DSMGDEF

The :GD tag is processed by the DSMGDEF APF which formats the glossary term and sets up the processing environment for the glossary definition with the following processing:

1. Checks if there is a glossary term defined. If there isn't, it means that there have been two :GD tags in a row with no intervening :GT tag. If this is the case a message is issued and a question mark is used as the term.

2. Skips a space.

3. Indents to where the term should be.

4. Begins the highlight font for glossary terms. This may be the default of 2 set in DSMPROF3 or it may have been overridden with the TERMHI attribute of the :GL tag.

5. Puts out the glossary term using literal mode just in case it happens to start with a period. The term is followed by a colon and a continuation character to cause it to be continued with whatever text follows it.

6. Restores the previous font.

7. Puts out a required blank to make sure that there is at least one blank between the colon and the following text. The required blank is put after the font is restored. This is done to avoid getting an underscored blank in the event that someone used an underscored font for highlighting the terms.

8. Sets the &@id@l symbol to null to indicate that the term has been fully processed.

DSMLPART

The :LP tag is processed by the DSMLPART APF which performs a skip and an indent (&@sk@l and &@in). Note that the &@in symbol is the amount of indention before the identifier, not the total indention to the text of the list items.

Ending Lists

DSMELIST

The DSMELIST APF ends lists, restores the environment if this is the end of all lists, remaps some tags, and resets nesting levels. DSMELIST processes the :EDL tag, the :EGL tag, the :EOL tag, the :ESL tag and the :EUL tag as follows:

1. Checks that a list is open. The &@nest@l symbol array contains an entry for each level of list that is open. If there are no elements in the array it means that there are no open lists which means that we don't need to be processing the DSMELIST APF. A message is issued if this is the case and the macro ends.

2. If closing the last list (&@nest@l(0) = 1):

   a. Resets the skip amount for lists to &@sk@ls. The &@sk@l symbol will contain the skip amount for the current list type. The default skip amount for lists is .75.

   b. Resets the highlight level for definition headings to its default value of &@hi@h which was defined in DSMPROF3.

   c. Skips a line at the end of the list.

   d. Indents &@in. The &@in symbol contains the base indention that was in effect before the current level of list was started.

   e. Resets &@nest@l array counter to zero and turns the &@nest@o and &@nest@u symbols off. These two symbols are used to keep track of the number of ordered and unordered lists that are open.

   f. Remaps the :LI, :LP, :DT, :DD, :DTHD, :DDHD, :GT and :GD tags to the invalid tag APF (DSM#CNTX) . This is done because these tags are not valid outside of a list.

3. If not closing the last list (&@nest@l(0) > 1):

   a. Adjusts the denesting symbols. The &@nest@l symbol counts the total number of lists that are open. For ordered and unordered lists, there are two separate symbols which keep track of these two types of lists. If we are ending an ordered or an unordered list the &@nest@&@ltype symbol will exist and we need to adjust it carefully using the denesting symbols (&@denest@&@ltype). The appropriate new value for the &@nest@&ltype symbol is a function of its current value and the denesting ring symbol. The &@denest@&@ltype symbol contains the numbers of the identifiers to use for each level of list. It acts as a ring, in that, when the level of nesting is deeper than the length of &@denest@&@ltype, the next identifier number is selected from the beginning of &@denest@&@ltype.

```
                          *-----------------*
                          |  *--------------+-*
                          V  V  V-----------+-+-*
if &@denest@&@ltype  =    6  1  2  3  4  5  | | |
                             A  A  A  A  A  | | |
   level 2 list restores ----*  |  |  |  |  | | |
   level 3 list restores -------*  |  |  |  | | |
   level 4 list restores ----------*  |  |  | | |
   level 5 list restores -------------*  |  | | |
   level 6 list restores ----------------*  | | |
   level 7 list restores -------------------* | |
   level 8 list restores ---------------------* |
   level 9 list restores -----------------------*
```

The denesting is done in this manner because the nesting is done in a similar manner. The identifiers wrap around such that the seventh level of list uses the same identifiers as the first level, and the eighth is the same as the second, and so on.

b. Skips a line to end the list.

c. Restores the previous list's formatting environment from &@nest@l. This includes the list type, the item counter, the indention amount, the incremental indention amount, the skip amount, the highlight fonts, and the break value. (&@ltype, &@item#, &@in, &@in@l, &@sk@l, &@hi@l, &@hi@hd, and &@break).

d. Decrements the list nesting level counter (&@nest@l(0)) by one because now one less level of list is open.

e. Indents for the level of list we are restarting (&@in) and the incremental value &@in@l.

f. Redefines the &@li@tab symbol to be the total indention (&$IN) that will be performed for this level of list. In other words, &@li@tab is set to where the text of the list item will start.

g. Resets the &@id@l to contain the symbol function needed to produce the list item identifier for ordered and unordered lists. We need to know the appropriate number of the identifier. This is calculated in &*a by selecting the nth character from the &@olistnest or &@ulistnest symbol, where n is the value of &@nest@o or &@nest@u calculated alone. The setting of &@id@l resolves like this:

```
.se @id@l '&V'&@id@l@&@ltype.&*a..'
.se @id@l '&V'&@id@l@o&*a..'
.se @id@l '&V'&@id@l@o2.'
```

h. Maps the appropriate tags for the list type to their respective APFs. For ordered, simple, and unordered lists, the :LI tag is mapped to the DSMLITEM APF. For definition lists, the :DTHD, :DDHD, :DT and :DD tags are mapped to the appropriate APFs. For glossary lists, the :GT and :GD tags are mapped to the DSMGTERM and DSMGDEF APFs.

DSM#RSET

The DSM#RSET macro is used to close open text structures for headings and document sections. This includes closing any lists that have been left open. This is done in the following manner:

1. Tests the list nesting counter, &@nest@l, for greater than zero.
2. Issues a warning message if a list is open.
3. Calls the DSMELIST APF to close list(s).

DSM#RSET is called by many macros which require that the &@state symbol have a value of "open." If there is an example, a figure, a title page or a footnote currently in progress, &@state will not have a value of "open."

Modifications to List Processing

Changing the List Item Identifiers

The list item identifiers to be used for ordered and unordered lists are defined in the profile. There are two parts to the definition—the selection of the actual identifier and the specification of the sequence in which they are to be used.

If you like the identifiers that have been selected but would like to change the order in which they are used for each level of nesting, the thing to do is change the value of &@olistnest and &@ulistnest. These two symbols are set to "123456" and "123." respectively. For page printers, &@ulistnest is changed to "12345" because there are many more interesting sorts of identifiers to choose from.

For ordered lists 9 different identifiers have been defined which have been assigned numbers from 1 to 9. The first six are used as the default identifiers for ordered lists.

LEVEL	DEFINED VARIABLE	EXAMPLE
1	`/&@item#..`	1.
2	`/&a'&@item#..`	a.
3	`/&@item#.)`	1)
4	`/&a'&@item#.)`	a)
5	`/&r'&@item#..`	i.
6	`/&r'&@item#.)`	i)
7	`/&R'&@item#..`	I.
8	`/&A'&@item#..`	A.
9	`/.dsm#supr &@item#`	(superscript 1)

Suppose you need to create outline style identifiers instead of the usual ones. All of the identifiers you need are already defined but you will need to respecify the order and identifier number in which they are used. This means changing the value of &@olistnest from "123456" to "785264." This will produce lists that are numbered like this:

```
I.  ...
    A.  ....
        i.  ...
            a.  ...
                i)  ...
                    a) ...
```

Note: You may also want to increase the value of the indention symbol (&@in@o) for ordered lists because of the greater number of characters in roman numerals. For example, 4 spaces are allotted to the identifier, but the roman number for 8 is "viii." When this is followed by a period or a parentheses, the identifier will extend too far to the right. This can be remedied by simply changing the value for &@in@o in the profile.

Suppose instead that you were only interested in three levels of nested ordered lists and that you wanted to number your lists enclosing the numbers in parentheses like this:

```
(1) ...
    (a) ...
        (i) ...
```

To do this you would need to change the value of &@olistnest to be "123" to specify that only three levels of identifiers are to be used. Then you would need to redefine the first three levels

```
.su off
.dv @id@l@o1 /(&@item#.)
.dv @id@l@o2 /(&a'&@item#.)
.dv @id@l@o3 /(&r'&@item#.)
.su on
```

You could, instead, specify that &@olistnest is "abc" and then define the identifiers as @id@l@a, @id@l@b, and @id@l@c. This would allow you to leave all of the original definitions in place. You could even switch back and forth between the two sets of identifiers by changing the value of &@olistnest within the document.

Changing Spacing and Indention Settings

Many of the starter set tags use spacing and indention values. For example, a space is skipped before and after lists and each level of list is indented 4 characters to the right of the previous one. The amount of vertical space skipped and the amount of indention are controlled by symbols set in DSMPROF3. By changing the value of the symbol you can change the amount of space or indention. The defaults are shown in Figure 5 on page 21. The symbols are set at the beginning of the profile using the .GS [GML Services] ARGS and VARS control words.

```
.gs args 10    2     4     4     0     3     4     4     2     0
.gs vars @in@d @in@f @in@z @in@o @in@p @in@q @in@s @in@u @in@x @in@g
.gs args .75   1     .75   .75   .75   1     .75   .75   1     .75
.gs vars @sk@d @sk@f @sk@z @sk@o @sk@p @sk@q @sk@s @sk@u @sk@x @sk@g
```

To change the amount of indention for any of the five types of lists simply change the value that is right above the appropriate symbol. The &@in@ symbols control the indention and the &@sk@ symbols control the spacing around the list. "d" stands for definition lists, "g" stands for glossary lists, "o" stands for ordered lists, "s" stands for simple lists, and "u" stands for unordered lists.

Changing the Highlight Defaults for Lists

A default highlight level is built into the profile for definition terms, glossary terms and definition headings. These can be changed by simply changing the settings of the symbols &@hi@h, &@hi@d and &@hi@g respectively to the new highlight level.

```
.gs args 1     3     2     2
.gs vars @sk@n @hi@h @hi@d @hi@g
```

So, for example, to make glossary terms appear in highlight level 1 instead of 2, change the last number on the .GS [GML Services] ARGS control word line to "1."

Using Decimal Notation for Ordered Lists

All of the list item identifiers that are used for ordered lists consist of a single number or character following by some punctuation characters. To create lists that are number in the following way,

```
1    Notice  that  the  first  level items are numbered with
     simple numbers.

     1.1   The second level items have decimal numbers.  No-
           tice that the indention keeps increasing for each
           level of list.

           1.1.1   The third level also has decimal numbers.

                   1.1.1.1   The fourth level is  just  more
                             of the same kind of thing.
```

Figure 20. **Decimal Ordered List Example:** This figure illustrates sample output after changing the list item identifiers to decimal format for ordered lists.

involves making several changes to the starter set macros and to DSMPROF3. This modification is more complicated that most of the other ones that are documented in this book because it requires changing some basic processing which has been built in to the starter set and it violates some of the assumptions that are built into list processing. We will see what these are as we go along developing the changes.

There are four separate areas of processing we need to change to created decimal lists:

1. We have to change the structure and definition of the list item identifier from a simple symbol to a symbol array so that it can hold the separate item numbers for each level simultaneously.

2. We also have to change the way in which the item identifier is incremented and printed to allow all the elements of the array to be printed separated by a period (.).

3. We also have to adjust the indention required for each level of list because the length of the identifiers will increase for each level of list.

Defining An Array for Item Numbers

Before delving into the changes that are detailed here, you might find it useful to review the information given in "Starter Set Initialization" on page 19 about how list item identifiers are defined in DSMPROF3 with the .DV [Define Variable] control word. What needs to be done to change the first level ordered list identifiers is to replace the following statement in DSMPROF3:

```
.su off
.dv @id@l@ol /&@item#..
.su on
```

with an identifier definition that will create an array. The easiest way to do this is:

```
.su off
.dv @id@l@ol /&@item#(*).
.su on
```

Associated with this change is an additional change which must be made to the setting for &@olistnest. Under normal circumstances, &@olistnest is "123456" indicating the identifier numbers for the first six levels of ordered list. The seventh level identifier wraps around and starts again using the same identifier as the first level list. With decimal numbering we can't let this happen. Several solutions are possible.

The easiest is to simply define a value of "111111" for &@olistnest.

```
.se @olistnest '111111
```

We have redefined only the first level identifier to be of the array format we want. Since it is an array, we want to use the same array for all levels of the list—just the elements of the array will vary.

As many 1's as you like can be put into &@olistnest, however, the number used will determine how many levels of list will be properly numbered, because the list processing code automatically wraps around when the level is greater than the number given in &@olistnest. Due to the large amount of indention which is created for nested decimal number lists, in single column format after about 5 levels of nested lists, the indention becomes wider than the page anyway, so the restriction to six levels of list is not too severe.

Note: Changing the automatic wrap around function for identifiers is not simple and cannot be done without affecting the identifiers for unordered lists also. The best way to handle this problem would be to put a test into DSM#LTYP to catch if the nesting level (&@nest@o) went over 6 and issue a message if so to warn users that the item numbers will be incorrect after that point.

Incrementing and Printing the Item Numbers

Now that the identifier is properly defined, we have to change the way in which it is processed in order to get the appropriate element in the array incremented and to get all the elements printed separated by periods. This involves changing the DSMLITEM APF to increment it properly. DSMLITEM currently contains the following line:

```
    .se @item# = &@item# + 1
```

which increments the item counter for the current level of list.

We need to replace this line with the following lines:

```
    .if &@ltype eq o
    .th .se @item#(&@nest@o.) = &@item#(&@nest@o.) + 1
```

The &@nest@o symbol will contain the nesting level of the current list, such that for the first level of list the first element of the &@item# array will be incremented, and the second element will be incremented for the second level of list, and so on.

So far we've redefined the identifiers and gotten them properly incremented for each item. The next step is to print the identifier, which currently is done by the following line of DSMLITEM:

```
   &@id@l.
```

The &@id@l symbol has a value of "&@item#(*)" due to the changes we made above to DSMPROF3. That's what we want. However, we need to adjust the array separator to be a period(.):

```
   .dc asep
   &@id@l.
   .dc asep
```

We also returned the array separator definition to its default value after we printed the identifier.

Now that we've got the item numbers correct for each level of the list, there's one additional thing we have to do to them. Since we're printing the whole array for each item, we have to get rid of the element which pertains to nesting levels we've closed. The logical place to do this is in the DSMELIST APF when we close the level.

When each level of list is started the &@item# number is saved. Since we've turned this symbol into an array, what actually gets saved will be the element index counter (that is, element 0 of the array). By reading in the element of the array in DSMELIST to restore the parameters for the previous level of list, we will automatically reset element zero of the array. However, when we print an array using (*), what we get is each element that exists, regardless of what the value of element zero is.

For example, when the fourth level list is ended, &@item#(0) is set back to 3 automatically. However, when we try to print &@item#(*) we get all four elements of the array. Therefore, we need to undefined or turn off the element of the array that refers to the level of list that is ended. When no previous list is being started, this is done as follows:

```
   .sk &@sk@l c
   .in &@in
   .if &@ltype eq o .se @item#(0) = 0
   .th .se @item#(1) off
   .se @nest@l(0) = 0
   ...
```

We have to decrement element zero of the list as well because the list control parameters are not saved and restored for first level lists.

When a prior list is being restarted (that is, we are ending a second level or higher list), we need to make the following adjustment:

```
.if &@ltype eq o .se @item#(&@item#(0).) off
.gs args &V'&@nest@l(&@nest@l(0).)
...
```

This line has to be added to DSMELIST *before* the list control symbols are restored from &@nest@l array.

Adjusting the Indention for List Items

The third part of the change to decimal ordered list involves adjusting the indention value for each level of list. The number for the first level is only two characters long (1.). For the second level it is three characters (1.1) and for the third it is five characters long (1.1.1). The indention needs to increase by two characters for each nested list.

The easiest way to approach this problem is to look at the incremental adjustment for each level of list as a separate value over and above the normal indentions applied to the list. The indention value for the list is contained in &@in@o which is defined in DSMPROF3 as 4. This value is moved into the &@in@l symbol when the list is started and it represents the incremental indention from where the identifier is placed to where the text starts.

If we were to create a separate indention symbol, named &@incr, to represent the incremental indention due to the identifier, it would need to be applied wherever we are currently applying the &@in@l indention. This means we have to:

1. Define and increment &@incr in DSM#LTYP.
2. Save and restore this value along with the other list control symbols.
3. Indent +&@incr in DSMLISTM to calculate the value of the &@li@tab symbol.
4. Indent +&@incr in DSMELIST when re-establishing a list.
5. Adjust the delayed indent set up in DSMLITEM for the second and subsequent lines of the item.

First, we'll define and increment the value in DSM#LTYP. For all list types except ordered list (o), we want the value of &@incr to be zero:

```
.se @ltype = substr 'zdosug' &*b 1
.se @incr = 0
.go list&@ltype
```

For ordered list, we want to set the value of &@incr to 1 for first level ordered lists and we want to increment it by 2 for each new level of list:

```
...listo
.if ...
.se &E'&@nest@o ...
.se *a ...
.if &@nest@o = 1 .se @incr = 1
.el .se @incr = &@nest@o * 2 - 2
```

However, we want the value to be 1 for a first level list, 2 for a second level list, 4 for the third level and so on. By taking the nesting level, multiplying it by 2 and subtracting 2 we get the right value.

Now that &@incr has the correct value, we have to save and restore it. The various list control symbols are saved in the &@nest@l symbol array by the DSMLISTM macro. These symbols are restored by the DSMELIST APF. Therefore, we have to add &@incr to the following line in DSMLISTM:

```
    .se *g = '&@ltype &@item# &@in &@incr
```

and to the following line in DSMELIST:

```
  .gs vars @ltype @item# @in @incr @in@l @sk@l @hi@l @hi@hd @break
```

We need to make this same adjustment to the DSMLQUOT and DSMELQU APFs because they also save and restore all these list control symbols. It doesn't matter where in the list of symbols you add &@incr. However, you must make sure that you always add it in the same location in all cases. In other words, if you put it in after the &@in symbol, as we did in the example above, then you must adjust all of the APFs to put it after &@in as these values are saved and restored positionally.

The next step is to apply the new incremental indention to the list. When the list is started in DSMLISTM, the total of the indentions for the previous level of list is saved in &@in which is the base indention value for the new level being started. The new incremental indention needs to be included in this calculation:

```
    .if &@nest@l(0) gt 1 .in &@in
    .th .in +&@in@l
    .th .in +&@incr
    .se @in = &DH'&$IN.dh
```

The indention values of the current level of list are also added up to calculate the value of &@li@tab. This symbol indicates where to insert space to after the identifier is placed to correctly position the first line of the list item text. The following lines need to change:

```
    .in &@in
    .in +&@in@l
    .in +&@incr
    .se @li@tab = &$IN
```

The &@li@tab symbol is used by the DSMLITEM APF.

When a list is restarted in DSMELIST, we also need to correct the indention setting by the value of &@incr:

```
    .in &@in
    .in +&@in@l
    .in +&@incr
```

The last adjustment involves correctly setting the delayed indention which controls the placement of the second and subsequent lines of the list item. This is calculated in the DSMLITEM APF and requires the following change:

```
    .in &@in
    .se *x = &dh'&@in@l + &dh'&@incr
    .in +&*x.dh after 1
```

In this case, the two incremental values are added together using device units to make sure they are in like units of space.

Examples and Figures

Example Processing

Examples are processed as keeps with formatting turned off. Various adjustments are also made to line spacing, word spacing, and fonts for page printers.

DSMXMP

The DSMXMP APF processes the :XMP tag by preparing the formatting environment for the example. It performs the following processing:

1. Checks that &@state is "open." If &@state is not "open," it issues a message and ends, ignoring the :XMP tag. The &@state symbol indicates whether there is a figure, footnote, example, or title page currently open. If there is, we can't process the example.
2. Sets &@state to "Exmpl" to indicate that an example is being processed.
3. Causes a break so that the text of the example will not be concatenated with the preceding text.
4. Saves the environment because we're going to make some changes to it and it will be easier to restore the environment than to reset everything we change.
5. Turns spelling verification, hyphenation, and formatting off.
6. Checks if a keep is in progress and if so, ends it. This is done to prevent the font stack from being lost. When we start the keep for the current example, any previous keep will be ended. Since the formatting environment (which includes the font save stack) is saved and restored around a keep, the font change that we are about to do will be lost from the stack by the restoration of the previous environment.
7. Starts the "xmpfont" font for page printers or restarts the current font (=) for all other devices.
8. Indents for examples (&@in@x). (Set in DSMPROF3 to 2.)
9. Skips for examples (&@sk@x). (Set in DSMPROF3 to 1.)
10. Starts a keep.
11. Resets vertical line spacing factors to 1.0 to prevent line spacing from being expanded by vertical justification.
12. Resets word spacing to its default value.
13. Resets extra spacing to its default value.
14. Processes the DEPTH attribute using the .SP [Space] control word. Remember that an Application Processing Function (APF) can be a macro, a symbol or a control word in SCRIPT/VS. In this case the value of the DEPTH attribute is simply a vertical space notation which needs to be issued with the .SP [Space] control word. The control word can be used directly rather than constructing a macro which would issue the control word.

DSMEXMP

The DSMEXMP APF processes the end tag for examples. It ends the example as follows:

1. Checks that &@state is "Exmpl." If &@state is not "Exmpl," it means that there is no example going now and this macro issues a message and ends, because there is nothing to be done.
2. Resets the &@state symbol to "open."
3. Ends the keep.
4. Restores the previous font. A real font change will only have occurred for the page printers. For all other devices, the current font will have been restarted.
5. Restores the previous environment.
6. Conditionally skips.

DSMPROF3

The profile maps the example tags to the APFs and defines the "xmpfont" font to be used for examples for page printers. The amount of skip before and after the example is set in &@sk@x to 1 and the amount of indention during the example is set in &@in@x to 2.

DSM#RSET

The DSM#RSET macro calls DSMEXMP if &@state is "Exmpl" indicating that an example is being processed.

Figure Processing

The various figure tags (:FIG, :FIGCAP, and :FIGDESC) are all processed by their own APFs. Each attribute also has its own processing macro.

The defaults for figures are established during initialization of the starter set.

DSMPROF3

The profile defines the amount of space to skip before and after the figure (&@sk@f) as 1. It also defines the amount of indention to be performed for the body of the figure (&@in@f) as 2.

The defaults for the PLACE and WIDTH attributes are established in DSMPROF3 as "top" and "page," respectively.

The fonts to be used for the figure caption and figure description for page printers are also defined in DSMPROF3.

Only the figure tag (:FIG) is mapped to its APF. The other figure tags (:FIGCAP and :FIGDESC) are mapped to their APFs when the figure is started.

DSM#SET

This macro, called during the initialization process by DSMPROF3, sets the figure number (&@fig#) to 1. The figure number is incremented by the figure caption processing in the DSMFCAP APF.

DSMFIG

The DSMFIG APF processes the :FIG tag by performing the following functions:

1. Checks that the &@state symbol is "open." If it isn't, a message is issued and the macro ends. A figure involves either a float or a keep structure. If &@state is not "open," it means that there is some other conflicting structure already going. To start the figure anyway would cause the user to get a SCRIPT/VS error message.

2. Sets &@state to "F" to indicate that a figure is being processed.

3. Executes a break so the text of the figure will not be concatenated with the preceding text.

4. Saves the current formatting environment. Because we are going to make some changes to the environment, it is easier to restore the environment than change it back.

5. Establishes the default parameters that control the formatting of the figure:

   a. &@figframe is set to "rule." This establishes the default frame for the figure. It can be overridden if the FRAME attribute is specified.

   b. &@fig@in is set to &@in@f which was set to 2 in DSMPROF3. This establishes the default indention for the body of the figure. The default can be changed by changing the value of &@in@f in DSMPROF3.

   c. &@figfo is set to off. This symbol is simply used as a flag within the figure macros. It is used by the DSMFDESC APF to tell if a figure caption has been processed.

   d. &@figtype is set to "fl" which stands for float. This symbol controls whether the figure is a keep or a float. The default is "fl" because the default placement of the figure is "top." The value of &@figtype can be changed if the PLACE attribute is specified. This symbol will be "issued" as it is expected to contain a control word.

   e. &@place is set to the value of &@figplace, which was set to "top" in DSMPROF3. The value of &@place can be changed if the PLACE attribute is specified. This symbol will be used as a parameter on the control word that was put into the &@figtype symbol (either "kp" or "fl").

   f. &@width is set to &@figwidth. The &@width symbol is also used on the .KP [Keep] or .FL [Float] control words. The &@figwidth symbol is set in DSMPROF3 to "page" and specifies the default width of the figure.

   g. &@efigpf is set to "off" to indicate that no font change has occurred yet for the figure. This symbol will be used to determine when to restore the previous font at the end of the figure.

   h. &@figcw is set up to issue a .HR [Horizontal Rule] control word. The default frame for a figure is a rule from left to right. The value of this symbol can be overridden if the FRAME attribute is specified. The @figrule on the .HR [Horizontal Rule] control word line is a rule name. The rule definition is in the profile but it has no parameters on it. This is done because the default we chose was the SCRIPT/VS default rule which is either .3mm thick for page printers or the current font for all other devices. The rule definition is provided in the profile and here on the .HR [Horizontal Rule] control word to make it easy for users to alter the rule for figures.

6. Processes the FRAME, WIDTH, and PLACE attributes, which may change the values of the &@figframe, &@place, and &@width symbols. See the descriptions of DSM@FRME, DSM@WIDT, and DSM@PLCE below for more details.

7. Resets left and right indention to zero.

8. Turns formatting and spelling verification off.

9. Changes the &@figtype symbol to "kp" if the &@place symbol is "inline" as a result of the PLACE attribute processing, because the figure will be formatted as an inline keep rather than as a float.

10. Starts single column mode with the .SC [Single Column Mode] control word if &@width equals "page" as a result of the WIDTH attribute.

11. Skips some space if the &@place symbol is "inline" to separate the figure from the text that surrounds it. The amount of space is in &@sk@f which is set in DSMPROF3 to 1.

12. Processes the control word contained in &@figtype with parameters of "on, &@place, &@width, and order." &@figtype will contain either "fl" (float) or "kp" (keep). &@place will be "top," "bottom," or null for inline figures &@width will be "page" or "column" depending on the WIDTH attribute.

    For example for a :FIG tag with no attributes specified this lines resolves as follows:

    ```
    .&@figtype on &@place &@width order
    .&@figtype on &@place page order
    .&@figtype on top page order
    .fl on top page order
    ```

13. For bottom figures (&@place is "bottom"), spaces &@sk@f. This is done because bottom figures need to be separated from the body text at the beginning of the figure. If the figure is to be placed at the top of the page, a skip will be performed at the end of the figure to separate it from the text which follows it.

14. Adjusts the word spacing and extra spacing values back to their default settings. The vertical justification factors on the .LS [Line Spacing] control word are reset to 1.0. We just want to be absolutely sure that the spacing within the body of the figure is not justified in any way at all.

15. Begins highlight font 2.

16. Draws a box top from left to right if &@figframe is "box." &@figframe will only be "box" if the FRAME attribute was specified with a value of "box."

17. Uses the &@figcw symbol to generate a frame if &@figframe exists and isn't "box" and if &@place is not "top." This symbol will either contain a split text control word (.SX [Split Text]) for character rules or will be a horizontal rule control word (.HR [Horizontal Rule]).

18. Returns to the previous font because only the frame is drawn in the highlight font.

19. Indents the proper amount for figures, &@in@f, which was set to 2 in DSMPROF3.

20. Indents right the same amount.

21. Sets the &@tg symbol to "f" to indicate to the DSM@IDS macro that an ID for a figure is being processed.

22. Processes the ID attribute using the DSM@IDS macro.

23. Processes the DEPTH attribute using the .SP [Space] control word.

24. Maps the :FIGCAP and :FIGDESC tags to their appropriate APFs.

| 25. Starts the "figfont" font for page printers or restarts the current font (=). In the starter set, the "figfont" font is not defined which means the body font will be used for the text of the figure. However, using the font in this macro makes it easy for you to specify a special font for use in figures by defining one with the .DF [Define Font] control word. The APFs will handle the font change.

26. Sets the &@efigpf symbol to "on" to indicate that a font change has been made.

DSMFCAP

The DSMFCAP APF processes the :FIGCAP tag. This APF numbers the figure for entry into the list of illustrations and for figure references. Therefore, if there is no :FIGCAP tag on the figure, the figure will not appear in the list of illustrations and cannot be referred to by the :FIGREF tag. The DSMFCAP APF performs the following processing:

1. Restores the previous font.
2. Sets the &@efigpf symbol to "off" to indicate that the font has been restored.
3. Scans to get the residual text to be used for the figure caption.
4. Indents left and right the appropriate amount (&@fig@in). &@fig@in is defined in the DSMFIG APF when the FIG tag is processed.
5. Turns formatting and spelling verification on. These had been turned off for the body of the figure.
6. Conditionally spaces 1 line.
7. Starts a special font for page printers named "figcap." For all other output devices, the current font is restarted.
8. Calculates, in device units, the proper amount of incremental indention for the second and subsequent lines of the caption and description. The width of the figure number, plus the width of the word "Figure" (&LL@F), plus three times the width of the character zero (&*w), plus the width of a period (&*period) is used.
9. Performs the amount of indention calculated above, plus the current indention.
10. Uses the amount of indention calculated as an undent to get the first line back out to where it should be relative to the left margin.
11. Formats the beginning of the figure caption which consists of: the word "Figure" from the &LL@F symbol, a required blank, and the figure number.
12. Inserts space to where the text of the caption should start and puts out the text of the caption itself.
13. Restores the previous font.
14. Makes an entry in the list of illustrations.

    a. Pads the figure number with a leading blank if the number is less than ten to make the numbers in the list of illustrations line up properly (at least up to figure 99).

    b. Defines a local symbol to contain "part" of a split text control.

    - Hex 00's are used as the delimiters in the .SX [Split Text] control word line.
    - The leftmost part of the split text line is made up of the word "Figure" (&LL@F), a required blank, the figure number (&@fig#), another required blank, and the text of the caption (&*line).
    - The middle portion of the split text line is a blank and a period which creates the dot leader.

    c. Defines a line of the #FIGLIST macro to contain a .OF [Offset] control word, a control word separator, and a .SX [Split Text] control word consisting of the local symbol defined above as the first part and the &@FN#&@fig# symbol as the second part.

    The #FIGLIST macro does not exist in the macro library for the starter set. It is dynamically constructed in the DSMFCAP APF as each figure caption is processed. The DSMFLIST APF, which is called to process the :FIGLIST tag, will invoke the #FIGLIST macro which will in turn produce the actual list of illustrations. See "Document Sections" on page 57 for more information about the DSMFLIST macro.

d. Saves the page number in the &@FN#&@fig# symbol. Ideally what we want to do is include the page number in the split text control word that we just entered into the #FIGLIST macro. However, we can't do this directly because the page number symbol on a .DM [Define Macro] control word line is not treated as a page number symbol and will remain as an ampersand.

   The .SE [Set Symbol] control word is special in that the page number symbol on the right hand side is recognized as the page number symbol and is replaced with the page number, which is what we want here. So we can capture the page number by setting a symbol to the page number symbol. The page number is remembered in a symbol whose name includes the figure number (&@FN&@fig#). This symbol name is what we used as the right hand side of the .SX [Split Text] control word that we put in the #FIGLIST macro above. The name of this symbol is unique because the symbol will not be resolved until the #FIGLIST macro is actually processed.

e. Increments the figure counter for the next figure.

f. Sets &@fig@fo to "on" to indicate to the DSMFDESC APF that a figure caption has been processed.

DSMFDESC

The DSMFDESC APF processes the :FIGDESC tag and formats the figure description. This macro performs different functions depending on whether or not a figure caption has been processed. The DSMFDESC APF functions as follows:

1. Checks to see if the formatting environment has already been set up by the DSMFCAP APF. The &@fig@fo symbol will be "on" only if there has been a figure caption.

2. If a caption was processed, DSMFDESC:

   a. Sets the &@fig@fo symbol to "off" for the next figure.
   b. Starts the "figcap" font for page printers or restarts the current font for all other devices.
   c. Puts out a continue control word, followed by a colon (:), followed by two required blanks, followed by a continuation character. This finishes off the caption.
   d. Restores the previous font.
   e. Starts the "figdesc" font for page printers or restarts the current font for all other devices.

3. If there was no figure caption, DSMFDESC:

   a. Restores the previous font.
   b. Sets the &@efigpf symbol to "off" to indicate that the font has been restored.
   c. Spaces 1 line.
   d. Resets left and right indention to &@fig@in which is 2.
   e. Turns formatting and spelling verification back on. These had been turned off in the DSMFIG APF when the :FIG tag was processed.
   f. Starts the "figdesc" font for page printers or restarts the current font for all other devices.

DSMEFIG

The DSMEFIG APF processes the end tag for figures (:EFIG). The figure is ended by DSMEFIG with the following processing:

1. Checks that a figure is being formatted by checking that &@state is "F." If &@state is not "F," a message is issued and the macro ends because there is nothing more to do.

2. Sets the &@state symbol to "open."

3. Restores the previous font only if the &@efigpf is "on." If there has been a figure caption or a figure description &@efigpf will be "off" indicating that the font has already been restored.

4. Resets left and right indention to zero.

5. Begins highlight font 2 in preparation for ending the figure frame.

6. Ends the box if &@figframe is "box."

7. If the frame is not a box but does exist, and the figure placement is not bottom, generates a frame by using the control word that is in the &@figcw symbol. The frame will be either a split text control word to generate a character frame or a horizontal rule control word.

8. Restores the previous font.

9. Skips the appropriate amount (&@sk@f) if &@place is "top."

10. Executes the control word in &@figtype with the parameter "off." &@figtype will be either "kp" or "fl" depending on whether the figure was inline or not.

11. Spaces conditionally if &@place is "inline."

12. Restores the formatting environment.

13. Remaps the :FIGCAP and :FIGDESC tags to the invalid tag APF, DSM#CNTX, because these tags are valid only within a figure.

DSM@FRME

The DSM@FRME macro processes the FRAME attribute of the :FIG tag. It is called by the DSMFIG APF if the FRAME attribute has been specified. The DSM@FRME macro performs the following functions:

1. Gets the first character of the parameter into a local symbol. The first character will be used to determine if the frame is to be omitted (N), a rule (R), or a box (B).

2. Saves the parameter in &@frame.

3. Sets up the &@figcw symbol to assume that a character frame is going to be generated using a .SX [Split Text] control word. The &@figcw symbol is used to actually generate the frame for the figure. If the &@figframe symbol ends up null, the .SX will be empty.

4. Sets the &@figframe symbol to null to generate an empty split text control if the first character is "N" for none.

5. Sets the &@figframe symbol to "rule" and resets &@figcw to a horizontal rule control word if the first character is "R" for rule.

6. Sets the &@figframe symbol to "box" if the first character is "B" for box. For figures with box frames, the indention to be applied to the body of the figure must not be allowed to be zero. If it is 0, it is reset to 2. If the indention were allowed to be zero, the figure text would be overlayed by the vertical rules of the box.

DSM@PLCE

The PLACE attribute of the :FIG tag is processed by the DSM@PLCE macro. It is called by the DSMFIG APF only if the PLACE attribute has been specified on the :FIG tag. The technique used in this macro is discussed in "Special Techniques" on page 13.

The default placement for a figure is "top" and is established in the profile, rather than in this macro. If the attribute value is not "top," "bottom," or "inline," the macro ends because these are the only values recognized.

The &@place symbol is set to either "top," "bottom," or "inline" depending on the value of the attribute.

DSM@WIDT

The DSM@WIDT macro processes the WIDTH attribute of the FIG tag. It is called by the DSMFIG APF. The technique used in this macro to validate the attribute value is discussed in detail in "Special Techniques" on page 13. The DSM@WIDT macro performs the following processing:

1. Checks to see if the uppercase of the attribute value is "PAGE" or "COLUMN." If so, the &@width symbol is set to lowercase "page" or "column" and the macro ends.
2. Assumes that the value has been given in horizontal space units if the value is neither place nor column.
3. Sets &@width to "page" if the width given is greater than the current column line length (&$CL); otherwise &@width is set to "column."

#FIGLIST

The #FIGLIST macro formats the list of illustrations. Only those figures that have figure captions will be listed, as these are the only ones that are numbered and labelled.

This macro is dynamically built by the DSMFCAP APF one line at a time for each entry. It is called by the DSMFLIST APF which processes the :FIGLIST tag.

It contains a .OF [Offset] control word and a .SX [Split Text] control word for each entry. The split text control word contains a symbol that resolves to the page number of the figure.

DSMFLIST

The DSMFLIST APF formats the list of illustrations. It processes the :FIGLIST tag. This macro is described in "Document Sections" on page 57.

DSM#RSET

The DSM#RSET macro calls DSMEFIG if &@state is "F" indicating that a figure is still being processed. This macro is described in "Miscellaneous" on page 163.

Modifications to Figures and Examples

Changing Figure Defaults

The :FIG tag has several defaults which are set in the profile.

The default placement for figures is "TOP." The default width of figures is "PAGE."

```
.gs args  123456      123         top        page
.gs vars  @olistnest @ulistnest  @figplace  @figwidth
```

To change these defaults, for example, to BOTTOM and COLUMN, just change the .GS [GML Services] ARGS control word to say "bottom" instead of "top" and "column" instead of "page." Make sure you specify these values are in lowercase because that's the way the figure macro is expecting them.

The font used for drawing the rules (or the weight of the rule for page printers) is controlled by the .DR [Define Rule] @figrule line in the profile. Changing this is discussed in "Modifying Starter Set Initialization" on page 36.

The default spacing around figures and the indention of the body of the figure is controlled by the value of &@sk@f and &@in@f which are set in the profile. How to change these is also discussed in "Modifying Starter Set Initialization" on page 36.

Moving the Figure Caption Outside the Frame

The style of figure chosen places the figure caption and description inside the figure frame. You might want to change this to put the caption and description outside the frame. The only thing that is tricky about ending the figure frame is whether to end it with the :FIGCAP tag, with the :FIGDESC tag (when there's no :FIGCAP), or with the :EFIG tag (when there's no :FIGCAP or :FIGDESC).

The easiest way to handle all three possibilities is to take the lines that end the frame out of DSMEFIG and put them in a separate new macro. Then use flags to tell when we want to end the frame:

1. Initialize a flag in the DSMFIG APF to show that the frame hasn't been ended yet.

   ```
   .se @endframe = off
   ```

2. Find the lines in DSMEFIG that end the frame. These are:

   ```
   .in
   .ir
   .bf hi2
   .if &@figframe eq box .bx off
   .th .go @frdone
   .if &@place ne bottom .an &E'&@figframe eq 1 .an /&@figframe ne /box
   .th &@figcw
   ...@frdone
   .pf
   ```

3. Remove these lines and put them into a new APF named DSM@END.

4. Add one more line to the end of the DSM@END APF which sets the flag to "on" to indicate we ended the frame.

   ```
   .se @endframe = on
   ```

5. Fix up the DSMEFIG APF to only call DSM@END when the flag is off.

   ```
   .if &@endframe eq off .dsm@end
   ```

   This test and possible call to DSM@END should go where the original lines were in DSMEFIG.

6. Fix up the DSMFCAP APF to call DSM@END *before* any other processing.

   ```
   .dsm@end
   ```

7. Fix up the DSMFDESC APF to call DSM@END only if there was no :FIGCAP tag. The DSMFDESC APF is actually divided into two parts. The first part is used when there has been a figure caption and the second part is used when there was no caption. The call to DSM@END belongs only in the second part because if there was a caption the frame will have already been ended.

Therefore, add

```
        .dsm@end
```

right after the label "format."

Now test it and make sure it works!

Changing the Example Defaults

The formatting of examples is controlled by several things that are set up in the profile. The &@sk@x and &@in@x symbols are defined in DSMPROF3 and control the amount of space skipped before and after the example and the amount of indention applied to the text of the example. Changing these values is described in "Modifying Starter Set Initialization" on page 36. We have also chosen to format the body of the example in a special font, "prestige elite." This font is defined with a name of "xmpfont" under the font definition section for the 4250 printer.

```
    .df xmpfont type('prestige elite'
```

If you wish to change the font used for examples simply change the font definition:

```
    .df xmpfont type('courier' expanded)
```

If you want examples set in the body font, simply delete the definition of the "xmpfont" font or comment it out:

```
    .*df xmpfont type('prestige elite')
```

For page printers, we have also chosen to format the body of the example in a special font—"prestige elite" for the 4250 printer and "prestige" for the IBM 3820 Page Printer and 3800 Printing Subsystem Model 3. These fonts are defined with a name of "xmpfont" in the profile under the font definition sections for the 4250 printer, 3800 Printing Subsystem Model 3, and IBM 3820 Page Printer.

```
    .df xmpfont type('prestige elite'
    .df xmpfont type('prestige'
```

If you wish to change the font used for examples simply change the font definitions.

```
    .df xmpfont type('courier' expanded)
    .df xmpfont type('courier')
```

If you want examples set in the body font, simply delete the definitions of the "xmpfont" font or comment them out.

Quotes, Notes, Footnotes and Highlights

Quote Processing

DSMPROF3

The profile defines various symbols for use in formatting quotations (:Q) and long quotations (:LQ). These include &@sk@q, the skip value before long quotations, and &@in@q, the indention for long quotations. It also defines the &@oquote symbol to contain the appropriate open quotation marks for each level of quotation. The &@cquote symbol contains the close quotation marks. These symbols have a special definition for page printers because there are typographical quotation marks available for these devices. This means that the quotation marks typed in directly by the user aren't the same characters that the quotation tags use. This can produce unusual looking results.

The lengths of the &@oquote and &@cquote symbol values place a restriction on the number of nested quotations you can have and still get quotation marks. However, because the limitation is 14 levels of quotations going at once, this is not a serious restriction for most users.

Additionally, four more symbols are defined:

```
&oq  - for single open quote marks
&oqq - for double open quote marks
&cq  - for single close quote marks
&cqq - for double close quote marks
```

These are provided for page printer users who wish to be able to make the correct typographical quotation marks throughout their documents.

The profile also maps the quotation tags to their respective APFs.

DSM#SET

The DSM#SET macro sets the footnote number to 1 during the initialization process.

DSMQUOTE

The :Q tag is processed by the DSMQUOTE APF which produces the appropriate opening quotation marks (single or double depending on the nesting level). The DSMQUOTE APF performs the following processing:

1. Substitution is turned off so that we can substring the quotation mark out of the &@oquote symbol.

2. The quotation nesting level is tracked in the &@nest@q symbol. This symbol is incremented each time a :Q tag is processed and is decremented by DSMEQUOT each time a :EQ tag is processed.

3. The &@oquote symbol, which is defined in DSMPROF3, contains a string of single and double quotation marks. The appropriate quotation mark is selected from &@oquote using the nesting level. For example, if &@nest@q is 1, the first character is selected from &@oquote and this is a double quotation mark. When the nesting level is 2, the second character is selected which is a single quotation mark.
4. Substitution is turned back on.
5. The quotation mark is inserted into the document.

DSMEQUOT

The end quotation tag (:EQ) is processed by the DSMEQUOT APF which performs the following functions:

1. If there are no quotations open, issues a message and the APF ends immediately because there is nothing to do.
2. Turns substitution off so that we can substring the quotation mark out of the &@cquote symbol.
3. Selects the proper closing quotation mark out of the &@cquote symbol using the nesting level (&@nest@q) to pick the correct one.
4. Decrements the nesting level symbol &@nest@q by one.
5. Turns substitution back on.
6. Inserts the ending quotation mark into the document.

DSMLQUOT

The DSMLQUOT APF processes the :LQ tag. Long quotations are processed very much as if they were a level of list nesting. This is because of the need to coordinate the indention between lists and long quotations as they can occur inside of each other. The following processing is performed:

1. Performs a conditional skip (&@sk@q).
2. Saves the current list control symbol values in the &@nest@l symbol array. These symbols are used for nesting and denesting lists and quotations. Some of the symbols are first put in the local &*h symbol simply because all of the symbol names won't fit on the next line which sets the next element of the &@nest@l symbol array. See the discussion of list control symbols in "List Processing" on page 97 for more details on the &@nest@l symbol.
3. Tests &@nest@l to determine if there are any open lists or open long quotations. The &@nest@l element counter is greater than one if there are any lists or quotations in progress. If the &@nest@l element counter is greater 1, we have to do a few things before we can set up the proper indention for the long quotation.

   The indention for each level of list and for long quotations is kept in two parts—the base indention prior to starting the list or quotation and the incremental indention (&@in@l) attributable to the list or quotation. To achieve the proper indention, both values must be used. The &@in symbol contains the amount of base indention for what is already open and &@in@l contains the incremental amount for each level of list (usually 4). Since we are starting a new level of indention for the long quotation, we need to get &@in and &@in@l incremented to indicate the appropriate values for the long quotation.

   The appropriate new value for the &@in symbol is the sum of the current value of &@in and &@in@l. The easiest way to get this sum is to actually perform the indention and use the value of &$IN. Then &$IN is used to reset the &@in symbol to the correct indention for the current long quotation.

4. Sets the control symbols for lists and long quotations as follows:

&@item#	This is the item counter. It is set to zero for long quotations.
&@ltype	This indicates the current type of list or quotation that is open. It is set to "q" to indicate a long quotation is in progress.
&@hi@l	This symbol controls the highlight level for the terms in the current list. For long quotations it's not needed so it is set to highlight level 0.
&@hi@hd	This symbol controls the highlight level for the headings in the current list. For long quotations it's not needed so it is set to highlight level 0.
&@in	This is the base indention value in effect prior to starting the long quotation. It is set to the value of &$IN calculated in device units[32].
&@in@l	This is the incremental indention value for the long quotation. It is set to the value of &@in@q calculated in device units[32]. &@in@q is defined as 3 in DSMPROF3.
&@break	This symbol indicates, when non-null, that the break option for definition lists has been specified. It is not used for processing long quotations.

5. Adjusts the long quotation formatting environment which consists of a left and right indention and possibly a font change. The left indention consists of the base indention (&@in) plus the incremental indention for long quotations (&@in@q). The right indention is also set to &@in@q which is set to 3 in the profile.

   A long quotation font, which is slightly smaller than the body font, exists only if we are formatting for page printers. For page printers the "lqfont" font is started, otherwise the current font is restarted.

DSMELQU

The DSMELQU APF processes the end tag for :LQ as follows:

1. Restores the previous font. The font is really only changed if we're formatting for a page printer. In other cases, the current font has been restarted and is now restored.

2. Performs a conditional skip (&@sk@q).

3. Decrements the right indention by the amount we incremented it when we started the long quotation (&@in@q).

4. Checks the &@nest@l element counter to make sure it indicates that a long quotation is open. If the &@nest@l element counter is zero, it means that no long quotations or lists are open and the macro ends because there is nothing more to be done.

5. If &@nest@l indicates that only the long quotation is open (that is, it is 1), ends the long quotation by:

   a. Setting the &@nest@l counter back to zero
   b. Restoring the left indention to what it was previously, which is in the &@in symbol.
   c. The macro ends.

6. If the long quotation occurred within either another long quotation or within a list, restores the environment to where we were before.

   a. The main list control symbol values are restored from the last element in the &@nest@l array.

[32] Device units have to be used here to avoid rounding errors when formatting for page printers.

b. The nesting level (&@nest@l(0)) is decremented by 1. It was incremented when we started the long quotation as these are kept track of very much like lists are. See the discussion of the list control symbols in "List Processing" on page 97 for more details on these symbols and what they do. The values being restored here were saved in DSMLQUOT when the long quotation was started. They are simply being restored here.

c. The &@id@l symbol, which contains the form for the list item identifier, is reset based on the level of nesting for the particular list type being restarted. For example, if the second level of unordered list is being started, the &@id@l symbol will be set as follows:

```
.se @id@l '&V'&@id@l@&@ltype.&*a..'
.se @id@l '&V'&@id@l@u&*a..'
.se @id@l '&V'&@id@l@u2.'
```

where &*a will be set to the level of list being started, &@ltype will be a single letter indicating the type of list. The value of &V'&@id@l@u2 will be a defined variable name whose value will be something like "&X'db." See the discussion of the defining the list item identifiers in "Starter Set Initialization" on page 19.

d. The correct left indention is restored using the &@in and &@in@l symbols. The &@in symbol contains the base indention in effect when the list was started and the &@in@l symbol contains the incremental indention attributable to the current list.

e. The &@li@tab symbol is restored to contain the current indention value for the list items. This symbol will be used for list items to insert space to where the text of the item should begin after the identifier is inserted.

DSM#RSET

This macro is used to ensure that there is no footnote, list, quotation, title page, figure, or example currently in progress. If there is, DSM#RSET calls the appropriate APF to end the structure. In the case of inline quotations, where &@nest@q is non-zero indicating that inline (short) quotations are open, DSM#RSET calls DSMEQUOT to close them. Long quotations are included in &@nest@l element counter which indicates the level of list nesting. If the &@nest@l element counter is non-zero, DSM#RSET calls DSMELIST which performs the necessary action to close the long quotation or list that is open.

Notes

DSMNOTE

The DSMNOTE APF processes the :NOTE tag as follows:

1. Skips as for a paragraph (&@sk@p).

2. Begins highlight level font 2.

3. Prints "Note:" followed by a colon.

4. Restores the previous font.

5. Produces the required blank. The required blank is added after the font has been restored to avoid getting an underscored blank in case the definition of the highlight level 2 font is changed to include underscoring.

Footnote Macros

DSMPROF3

The profile, DSMPROF3, initializes the following values for footnotes:

1. Sets &@sk@n to 1. This symbol determines the amount of white space preceding the footnote.
2. Defines a rule named "@fnldr" to be used for the footnote leader.
3. Sets &@fnldrlen to 16. This symbol determines the length of the footnote leader rule.
4. Determines the style of superscripting based on the physical device being used (&$PDEV) and puts it in the &@suprstyl symbol. The three styles supported are:
   - "parens" to produce parentheses,
   - "shifts" to use a small font shifted up, and
   - "nums" to use the available superscript numbers.
5. Maps the footnote tags to their respective APFs.

DSM#RSET

The DSM#RSET macro is used to ensure that there is no footnote, list, quotation, title page, figure, or example currently in progress. If there is, it calls the appropriate APF to end the structure. In the case of footnotes, where &@state is found to be "N," it calls DSMEFTNT.

DSM#STYL

The page layout style, which effects the way footnotes are formatted, is established by the DSM#STYL macro. For footnotes, this macro performs the following actions:

1. Sets &@fn1 and &@fn2 (left and right indention amounts for footnotes) depending on column layout:
   - For one and two column formats, &@fn1 and &@fn2 are both set to zero.
   - For offset style, &@fn1 is set to the beginning column position so that the footnote will line up with the offset text of the page and &@fn2 is set to 0.
2. Defines the footnote leader for all formats by:
   a. Spacing 1 line.
   b. Drawing a horizontal rule using the @fnldr rule defined in DSMPROF3. The rule starts at the left indention (&@fn1) as calculated above, and extends for &@fnldrlen, which is defined in DSMPROF3 to be 16.

DSMFTNT

The :FN tag is processed by the DSMFTNT APF in the following way:

1. Checks that &@state is "open" indicating that there are no figures or examples under construction. If there are, a message is issued and the APF ends immediately. We can't process a footnote while we're inside a figure or example because these are keeps or floats. To do so

would cause a SCRIPT/VS error message about keeps, floats and footnotes being mutually exclusive.

2. Sets &@state to "N" to indicate a footnote is in progress.

3. Checks the attribute stack to see if an ID attribute was specified.

4. If none was given, assumes that the footnote reference goes right here in the text and the DSM#SUPR macro is called with the footnote number as a parameter. This produces the footnote reference in the text.

5. Issues the .FN [Footnote] ON control word to start the footnote.

6. Spaces &@sk@n. This symbol is set in DSMPROF3 to 1.

7. Starts the footnote font ("fnt") or restarts the current font. The footnote font, "fnt," exists only for page printers.

8. Sets left indention for the footnote number to &@fn1. This symbol is set by DSM#STYL based on the column layout being used. It is zero for one- and two-column layouts and approximately 13 for offset style.[33]

9. Sets the &@tg symbol to "n" to indicate to the DSM@IDS macro that a footnote ID is being processed.

10. If an ID attribute was specified, the DSM@IDS macro is called to process it. See "Cross-References" on page 147 for details on ID processing.

11. Sets up the incremental left indention and the right indention for the footnote text:

    a. If the superscripts are going to be produced by changing to a smaller font and shifting the baseline up, the calculations must be done in the smaller font. This style of superscript processing is indicated by the &@suprstyl symbol which will be "shifts." The "super" font is started for page printers and the current font is restarted for all other devices.

    b. The &@fnis symbol is used in the DSM#SUPR macro to insert space to the location of the beginning of the footnote text. Its value is based on the current indention (&$IN) plus four figure spaces, calculated in device units. The existence of this symbol, &@fnis, is also used by the DSM#SUPR macro to tell whether it is creating a footnote reference or a footnote number.

    c. Right indention is set to the value of &@fn2 which is set by the DSM#STYL macro. This will be zero for all column styles.

    d. The next step is to calculate the proper indention for the second and subsequent lines of the footnotes. This is calculated to be four figure spaces in the superscript fonts.

    e. The previous font is then restored.

    f. A delayed incremental indention is set to the value that was calculated above. This will control the placement of second and subsequent lines of the footnote.

    g. The DSM#SUPR macro is called with the footnote number as a parameter.

    h. The footnote number (&@fn#) is incremented by 1.

    i. The &@fnis symbol is undefined because we no longer need it. This symbol is used by DSM#SUPR to tell if a footnote reference or a footnote number is to be created. We need it to be undefined so that subsequent calls to DSM#SUPR by DSMFNREF will be handled correctly.

[33] The exact amount varies by device type.

DSM#SUPR

The DSM#SUPR macro produces both the superscript footnote reference and the footnote number. It is called by both DSMFNREF and DSMFTNT with a footnote number as a parameter.

There are three styles of superscription supported by the starter set:

- For terminals, no real superscripts are available so the best we can do is enclose the number in parentheses. This style is labelled "parens."
- For 1403 and 3800 Model 1 output, real superscript numbers are used. This style is labelled "nums."
- For page printers, superscripts need to be constructed by changing to a small font and shifting the baseline up. This style is labelled "shifts."

The superscript style to be used is determined in DSMPROF3 based on the physical device (&$PDEV) being used. It is stored in the &@suprstyl symbol.

The DSM#SUPR macro performs the following processing:

1. Substitution is turned off.
2. The appropriate section of the macro is branched to to produce the style of superscript specified in &@superstyl. This can be "nums," "parens," or "shifts" depending on the device.

nums For the 1403 and 3800 Model 1, each digit of the footnote number is converted to the appropriate hexadecimal code to print the number as a superscript. In other words, a "1" becomes a hex "b1," a "2" becomes a hex "b2," and so on. This is done using the .TR [Translate Character] control word.

1. Defines the translations to convert numbers to superscripts.
2. Formats the number.
3. Cancels the previously defined translations.
4. Checks the existence of &@fnis. If this symbol exists, it indicates that we are producing a footnote number rather than a footnote reference. In that case, we need to insert some space after the number to position to where the text should start. The position is in &@fnis so all we have to do is a .IS [Inline Space] control word.

parens For terminals, the best we can do for superscripting is put the number in parentheses. This is done as follows:

1. Symbol substitution is turned back on.
2. If the &@fnis symbols does not exist it means that we are generating a footnote reference number. In that case, we simply put the parameter back out with parentheses around it and end the macro.
3. If &@fnis does exist, we are generating a footnote number and we don't use the parentheses. Literal mode is used just to be safe.
4. We then insert space to where the text should start, and end the macro.

| **shifts** Creating superscripts for page devices is a little more complicated. We need to change fonts and shift the baseline up to print a smaller font at the top of the line so that the top of the number lines up with the top of the normal letters on the line. The calculations are as follows:

1. Substitution is turned back on.
2. First, we get the vertical height of the current font. This is 1 vertical em-space.

3. Then we switch to the superscript font ("super") which was defined in the profile as a 6 point font.

   If the "super" font is not defined, we restart the current font. This sequence can only occur if the profile has been modified to not define a font named "super" or has used the "shifts" superscript style for a device that doesn't have a "super" font defined.

4. We then get the vertical height of the characters in that font.

5. The difference between the two heights is the amount we will need to shift the baseline up.

6. The difference between the two heights could be zero or less than zero. This can happen only if the font definitions have been changed. However, it makes proceeding difficult because we are no longer working with a superscript font that is smaller than the normal font. In this case, we

   a. Restore the previous font
   b. Produce the footnote number in parentheses
   c. End the APF.

7. If the &@fnis symbol does not exist, it means we are producing a footnote reference rather than a footnote number. In that case, we

   a. Shift the baseline up the amount we calculated above.
   b. Produce the number continuing it with what preceding.
   c. Shift the baseline down the amount we shifted it up. We don't simply restore the baseline because it is possible that the baseline was not at 0 when we shifted it up.
   d. Restore the previous font.
   e. End the APF.

8. If the &@fnis symbol does exist we are producing a footnote number. This is done in the following way:

   a. Shift the baseline up the amount we calculated above.
   b. Produce the number in literal mode. Literal mode is used here to prevent the characters being formatted as the superscript from being interpreted by SCRIPT/VS as control characters of any sort.
   c. Shift the baseline down the amount we shifted it up. We don't simply restore the baseline because it is possible that the baseline was not at 0 when we started.
   d. Restore the previous font.
   e. Insert the required space to position to where the text of footnote should start. The position is in the &@fnis symbol which was set by the DSMFTNT macro.

DSMEFTNT

The DSMEFTNT APF ends formatting of a footnote when it processes the :EFN tag in the following way:

1. Checks that &@state is "N" indicating that a footnote is in progress. Issues a message and ends immediately, if &@state is not "N."

2. Sets &@state to "open."

3. Restores the previous font because a font change to either a smaller font or the current font has been done in the DSMFTNT APF.

4. Uses the .FN [Footnote] OFF control word to end the footnote definition.

Highlights

DSMPROF3 defines all of the fonts used in the starter set, including the highlighting fonts which are named hi0, hi1, hi2, and hi3. The definition used depends on the device being used. In the case of page printers, the highlight font definitions are all "type" defined fonts, meaning they inherit some characteristics from the current font at the time they are used. For example, highlight level 2 is defined as bold for page printers. In a footnote, which is in 9-point, we get a 9-point bold font when we use highlight level 2, but in the body of the document we get 10-point bold.

Because highlight level 2 always changes to the bold version of the current font, and it is possible that there is no bold version available, there are 3 alternate highlight fonts defined. The alternate highlights (althi1, althi2, and althi3) use only uppercase and underscore definitions that will always work. They are used whenever the three main highlight fonts are not available.

DSMCIT

The :CIT tag is processed by the DSMCIT APF. This macro starts highlight level font 1 or alternative highlight level font 1 if highlight level 1 font is not isn't available.

DSMECIT

The :ECIT tag is processed by the DSMECIT APF which ends highlighting by restoring the previous font.

DSMHP0

The DSMHP0 APF begins the highlight level 0 font for the :HP0 tag. There is no alternate highlight font for highlight level zero because the "hi0" font is defined to be the normal body font which must always exists.

DSMHP1

The DSMHP1 APF processes the :HP1 tag by beginning the highlight level 1 font or the alternate highlight level 1 font if the highlight level 1 font is not available.

DSMHP2

The DSMHP2 tag processes the :HP2 tag by beginning the highlight level 2 font or the alternate highlight level 2 font if the highlight level 2 font is not available.

DSMHP3

The DSMHP3 tag processes the :HP3 tag by beginning the highlight level 3 font or the alternate highlight level 3 font if the highlight level 3 font is not available.

DSMEHP

The DSMEHP APF, which processes the :EHP0, :EHP1, :EHP2, and :EHP3 tags, ends highlighting by restoring the previous font.

Modifications to Quotes, Notes, Footnotes and Highlights

Changing the Footnote Leader

The footnote leader is defined in the DSM#STYL macro which is called from DSMPROF3 during initialization. Because the DSM#STYL macro is also called when changing document sections (such as going from front matter to body) it is difficult to override the footnote leader from within a document. It really needs to be changed in the DSM#STYL macro. For this reason, the symbols for the most important footnote leader parameters are defined in DSMPROF3 to make it easier to change them.

To change the length of the footnote leader horizontal rule, change the value of &@fnldrlen in DSMPROF3:

```
.se @fnldrlen = 16
```

To change the size of the rule used, change the definition of the @fnldr rule that is in the profile:

```
.dr @fnldr w .2mm
```

To completely change the leader, we will have to modify the DSM#STYL macro. For example, if we wanted to skip 2 lines instead of 1 and use a row of asterisks, we would need to change the DSM#STYL macro to contain the following:

```
...fnldr
.fn leader
.sp 1
**********
.fn off
```

Printing Footnotes at the End of a Chapter

Normally, footnotes are printed on the page on which they occur. Some applications, however, require the printing of footnotes to be deferred until the end of a chapter. This can be accomplished by putting footnotes into a named section area and placing that area at the end of each chapter.

The .DA [Define Area] control word can be used to define an area in which we can collect the footnotes. An area is simply a column that can be placed anywhere on the page. See the chapter entitled "Placing Text in Named Areas" in *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for more information on areas. The following control word should be placed into DSMPROF3:

```
.da fnote@a 0 section width &dh'&$LL..dh
```

The above .DA [Define Area] control word defines a section area named *fnote@a.* We did not specify a horizontal displacement for this area, so it will be placed starting at the left margin. We have specified the width to be the current line length as defined with the .LL [Line Length] control word. If we are in a two column format, we want the footnotes to extent across both columns, not just the first one.

We also need to modify the DSMFTNT APF to collect the footnotes in the area. The following line should replace the .FN [Footnote] ON control word in the DSMFTNT APF:

```
.ar fnote@a on
```

We also need to modify the DSMEFTNT APF to close the area at the end of the footnote text. The following control word should replace the .FN [Footnote] OFF control word in the DSMEFTNT APF:

```
.ar off
```

The above two changes will cause the footnotes to be collected in an area. The next step is to get the area placed at the end of the chapter.

How do we know when we are at the end of a chapter? A head level 0 or head level 1 indicates the end of a chapter and the beginning of a new one. The DSM#DUPL macro is called at the start of the DSMHEAD0 and DSMHEAD1 APFs when the next chapter or part is started. If there are footnotes to be placed, we need to eject to a new page and then print the footnotes. The &ad' attribute can be used to determine if the area contains any text. We may also want to put out a heading for the footnotes. We will assume the text for the heading is in a symbol named &LL@Ftnt. Here's what the DSM#DUPL macro looks like when we are through:

```
.if &$PN eq 0 .me
.fl dump
.if &ad'fnote@a ne 0
.th .pa nostart
.th .h2 &LL@Ftnt
.th .ar put
.* DUPLEX - EJECT TO EVEN PAGE. NOT DUPLEX - REMOVE THE EJECT LINE   *
.dm dsm#dupl(&$LNUM.) off &$CW..se *a = &$LNUM + 20
.dm dsm#dupl(&$LNUM.) off &$CW..if yes eq no .dm dsm#dupl(&*a.) off
.if SYSPAGE eq ODD .pa
.pa nostart
```

We need to make sure the footnotes for the last chapter of the document are also printed. We can accomplish this by placing the same control words we added to the DSM#DUPL macro at the beginning of the epifile in DSMPROF3. Here's what the epifile in DSMPROF3 looks like when we are through:

```
.if &ad'fnote@a ne 0
.th .pa nostart
.th .h2 &LL@Ftnt
.th .ar put
.if &E'&SYSVARW ne 0 .an &@lastpass eq yes .dsm#writ
.if &SYSVARX eq yes .an &@lastpass eq yes .dsm#xlst
```

We also need to define the symbol &LL@Ftnt in the DSM#SETS macro. This symbol is used in the head level 2 that prints out at the top of the footnote page. The following line needs to be added to the DSM#SETS macro:

```
.se LL@Ftnt 'Footnotes
```

Using a Hanging Indent for Notes

The :NOTE tag does not normally perform any indention at all. To indent notes to look like this:

Note: This is what a hanging note would look like if we made the following change to the DSMNOTE APF.

Some changes need to made to the DSMNOTE APF.

The DSMNOTE APF looks like this:

```
.sk &@sk@p
.bf hi2
&LL@Note.:&$CONT
.pf
&$RB.&$CONT
```

To change this to produce a hanging indent, all we need to do is add an incremental indention delayed for one line. The indention needs to be incremental because we don't want to be concerned with whether or not there is any indention at the time of the note. The indention is delayed for one line so that the "NOTE:" line will not be indented. The amount of the indention must be calculated as the width of the word "Note," the colon, and the required blank.

To calculate the indention, we must first define a symbol to be a colon. We have to do this because using a colon directly in the width calculation doesn't work because it's a special character. Then we simply add up the device units of the width of the various pieces and use this value in the .IN [Indent] control word.

```
.sk &@sk@p
.bf hi2
.se *c = :
.se @in@note = &DH'&W'&LL@Note + &DH'&W'&*c + &DH'&W'&$RB
.in +&@in@note.dh after 1
&LL@Note.:&$CONT
.pf
&$RB.&$CONT
```

This change to the DSMNOTE APF will causes the second and subsequent lines of the note text to be indented the proper amount. However, it is now necessary to create an end tag for :NOTE in order to cancel the indention for the next text item is processed.

This means the .AA [Associate APF] control word line in DSMPROF3 for the :NOTE tag needs to be changed to include the name of an APF to process the end tag. Let's assume the APF will be named "ENDNOTE." The new .AA [Associate APF] line is shown below:

```
.aa note dsmnote (noatt) endnote
```

The ENDNOTE APF simply needs to end the indention.

```
.in -&@in@note.dh
```

Note that we have decremented the indention rather than reset it to zero. This is because we don't know whether it was zero when we started or not. If we decrement it the same amount we incremented it in the start tag then we don't have to be concerned with any indention which may have been going at the time of the :NOTE tag.

Indexing

Index Tag Macros

The processing of ID attributes and cross references for the index tags is slightly different from other types of cross references. Therefore, it is discussed here, rather than in the chapter on cross referencing. The cross referencing of index ids is an important part of processing the index tags themselves. For other types of cross referencing, the cross reference capability is separate from the tags themselves and therefore the discussions can be separated.

The tag to APF mappings for all of the index tags is shown in Figure 8 on page 29. These APFs are discussed in detail here.

DSMINDEX

The DSMINDEX APF processes the :INDEX tag and formats the index in the following manner:

1. Calls the DSM#RSET macro to end any open lists, footnotes, and so on. (See "Miscellaneous" on page 163.)
2. Calls the DSM#DUPL macro to get to the beginning of the next odd page if duplexing is active. (See "Miscellaneous" on page 163.)
3. Sets &@shead to the &LL@Index symbol whose value is "Index." The &@shead symbol is used in the running footing.
4. Defines an IEH macro to process the internally generated .IE [Index Entry] H control words. All the IEH macro does is call the DSMIEH macro passing along the control word parameters.
5. Loads the GML index header macro (DSMIEH) from the library. This is done because we are going to turn the library search for macros off entirely. If we didn't explicitly tell SCRIPT/VS that the DSMIEH macro was in the library, SCRIPT/VS wouldn't be able to find the macro.
6. Saves the current formatting environment because we're going to make some changes to it and it will be easier to restore the environment than change it back.
7. Calls DSM#STYL to get a two column layout.
8. Turns formatting style to "left" as we don't want any horizontal justification to occur within the index.
9. Turns library look-up for macros off. This is done for performance reasons. Searching the library for symbols and macros is very costly, if it turns out to be a control word, as it usually is with index processing. Since we know there are no symbols that need to be "fetched" for index processing, we simply turn it off.

10. Uses the .IX [Index] control word to format the index. The title of the index is obtained from the &LL@Index symbol which is defined in the DSM#SETS macro during initialization.

11. Enables the library search for macros after the index is completely formatted.

12. Restores the previous environment.

DSMINDX1

DSMINDX1 processes the :I1 tag in the following manner:

1. Scans to get the residual text for the tag into &@it1.

2. Sets &@ilevel to 1 to indicate to the attribute processing macros that a first level index entry tag is being processed.

3. Sets &@it2 and &@it3 to null.

4. Sets &*t4 to the page number symbol, &$PS. This is how the page number gets into the index. The user can specify his own fourth level entry to be something other than page number, by using the PAGEREF attribute.

5. Sets &@tg to "i" to indicate to the DSM@IDS macro that an id for an index entry tag is being processed.

6. Processes the PG or PAGEREF attribute with DSM@PGRF. This may result in &*t4, the fourth index term, being redefined to something other than the page number symbol. It may also cause the &*x symbol to be set to "start," "order" or "end."

7. Processes the ID attribute with DSM@IDS.

8. If there was no text on the tag (&@it1 has a length of 0), issues an error message and no index entry is created.

9. If the entry exists, sets up the .PI [Put Index] control word such that &*x can provide additional control word parameters. See "DSM@PGRF" on page 144 for details. The hexadecimal "01"s are the delimiters on the .PI [Put Index] control word line and the index term itself is in &@it1.

DSMINDX2

DSMINDX2 processes the :I2 tag in the following manner:

1. Moves the current first level index term from &@it1 to &#it1

2. Scans to get the residual text for the tag into &@it2.

3. Sets &@ilevel to 2 to indicate to the attribute processing macros that a second level index entry tag is being processed.

4. Sets &@it3 to null.

5. Sets &*t4 to the page number symbol, &$PS. This is how the page number gets into the index. The user can specify his own fourth level entry to be something other than page number, by using the PAGEREF attribute.

6. Sets &@tg to "i" to indicate to the DSM@IDS macro that an id for an index entry tag is being processed.

7. Processes PG or PAGEREF with DSM@PGRF. This may result in &*t4, the fourth index term, being redefined to something other than the page number symbol. It may also cause the &*x symbol to be set to "start," "order" or "end."

8. Processes the REFID attribute using the DSM@RIDI macro. This attribute process may change the first level term.

9. Processes the ID attribute with the DSM@IDS macro. ID processing is described in detail in "Cross-References" on page 147.

10. If there was no text on the tag (&@it2 has a length of 0), or the first term (&@it1) is missing, issues an error message and no index entry is created.

11. If the entry exists, sets up the .PI [Put Index] control word such that &*x may contain additional control word parameters. The hexadecimal "01"s are the delimiters on the .PI [Put Index] control word line and the index term itself is in &@it2. The first index term is in &#it1 and the fourth term or the page number symbol is in &*t4.

DSMINDX3

DSMINDX3 processes the :I3 tag in the following manner:

1. Moves the current first level index term from &@it1 to &#it1 and moves the current second level index term from &@it2 to &#it2. These two symbols, &#it1 and &#it2 are the ones actually used in generating the index entry. If there is a REFID attribute on the :I3 tag, it may override those two symbols by linking the I3 term to a different set of level 1 and 2 terms.

2. Scans to get the residual text for the tag into &@it3.

3. Sets &@ilevel to 3 to indicate to the attribute processing macros that a third level index entry tag is being processed.

4. Sets &*t4 to the page number symbol, &$PS. This is how the page number gets into the index. The user can specify his own fourth level entry to something other than page number, by using the PAGEREF attribute.

5. Processes PG or PAGEREF with DSM@PGRF.This may result in &*t4, the fourth index term, being redefined to something other than the page number symbol. It may also cause the &*x symbol to be set to "start," "order" or "end."

6. Sets &@tg to "i" to indicate to the DSM@IDS macro that an id for an index entry tag is being processed.

7. Processes the REFID attribute using the DSM@RIDI macro. This attribute process may change the first and second level terms.

8. Processes the ID attribute with the DSM@IDS macro. See "Cross-References" on page 147 for details on this macro.

9. If there was no text on the tag (&@it3 has a length of 0), or the first term (&#it1) or the second level term (&#it2) are missing, issues an error message and no index entry is created.

10. If the entry exists, sets up the .PI [Put Index] control word such that &*x may contain additional control word parameters. The hexadecimal "01"s are the delimiters on the .PI [Put Index] control word line and the index term itself is in &@it3. The first index term is in &#it1. The second term is in &#it2 and the fourth term or the page number symbol is in &*t4.

DSMIHD1

The DSMIHD1 APF creates a primary index heading entry when it processes the :IH1 tag as follows:

1. Gets the residual text into &@it1 and saves it in &#it1. The &#it1 symbol is the one that will actually be used to generate the index control word line.

2. Sets &@ilevel to 1. This is done to indicate to the attribute processing macros that the tag being processed is a first level entry.

3. Sets &@it2 and &@it3 to null.

4. Sets &@tg to "i" to indicate the DSM@IDS macro that an id for an index entry is being processed.

5. Processes the PRINT attribute with the DSM@IPRT macro.

6. Processes the ID attribute with the DSM@IDS macro. See "Cross-References" on page 147 for details on this macro.

7. Processes the SEE attribute with the DSM@SEE macro.

8. Processes the SEEID attribute with the DSM@SEEI macro.

9. Tests the existence of &*r. The &*r local symbol is set if either the DSM@SEE macro or the DSM@SEEI macro is processed. If it does exist, &*x is set to "ref" to generate an index reference rather than a real index entry. This is the .PI [Put Index] control word parameter that suppresses the page number.

10. If there was no text on the tag (&@it1 has a length of 0), issues an error message and no index entry is created.

11. If there was text on the tag, sets up the .PI [Put Index] control word using &*k which may have been set by the DSM@IPRT macro to provide a sort key parameter. The hexadecimal "01"s are the delimiters on the .PI [Put Index] control word line and the index term itself is in &#it1. The &*r symbol may contain an index reference in which case *x has been set to "ref." This combination creates the "See" and "See also" references.

DSMIHD2

The DSMIHD2 APF creates a second level index heading entry. when it processes the :IH2 tag as follows:

1. Gets the residual text into &@it2 and saves it in &#it2. The &#it2 symbol is the one that will actually be used to generate the index control word line.

2. Sets &@ilevel to 2. This is done to indicate to the attribute processing macros that the tag being processed is a second level entry.

3. Sets the &@it3 symbol to null.

4. Sets &@tg to "i" to indicate to the DSM@IDS macro that an id for an index tag is being processed.

5. Processes the PRINT attribute with the DSM@IPRT macro.

6. Processes the ID attribute with the DSM@IDS macro. See "Cross-References" on page 147 for details on this macro.

7. Processes the SEE attribute with the DSM@SEE macro.

8. Processes the SEEID attribute with the DSM@SEEI macro.

9. Tests the existence of &*r which is set if either the DSM@SEE macro or the DSM@SEEI macro has been used. If it does exist, &*x is set to "ref" to generate an index reference rather than a real index entry.

10. If there was no text on the tag (&@it2 has a length of 0), or there is no active first level entry (&@it1 has a length of 0), issues an error message and no index entry is created.

11. If the entries exist, sets up the .PI [Put Index] control word where &*k may have been set by the DSM@IPRT macro to provide a sort key parameter. The hexadecimal "01"s are the

delimiters on the .PI [Put Index] control word line and the index term itself is in &#it2. The first level index term is in &@itl. The &*r symbol may contain an index reference in which case *x has been set to "ref." This combination creates the "See" and "See also" references.

DSMIHD3

The DSMIHD3 APF creates a third level index heading entry. when it processes the :IH3 tag as follows:

1. Gets the residual text into &@it3 and saves it in &#it3. The &#it3 symbol is the one that will actually be used to generate the index control word line. The primary and secondary terms are in &@it1 and &@it2.
2. Sets &@ilevel to 3. This is done to indicate to the attribute processing macros that the tag being processed is a third level entry.
3. Processes the PRINT attribute with the DSM@IPRT macro.
4. Sets &@tg to "i" to indicate to the DSM@IDS macro that an id for an index tag is being processed.
5. Processes the ID attribute with the DSM@IDS macro. See "Cross-References" on page 147 for details on this macro.
6. If there was no text on the tag (&@it3 has a length of 0), or there is no active first level entry (&@it1 has a length of 0), or there is no active second level entry (&@it2 has a length of 0), issues an error message and no index entry is created.
7. If the entries exist, sets up the .PI [Put Index] control word where &*k may have been set by the DSM@IPRT macro to provide a sort key parameter. The hexadecimal "01"s are the delimiters on the .PI [Put Index] control word line and the index term itself is in &#it3. The first level index term is in &@itl and the second level index term is in &@it2.

DSMIREF

The :IREF tag is processed by the DSMIREF APF as follows:

1. Sets a local symbol, &*t4, to the page number symbol (&$PS).
2. Processes the REFID attribute using the DSM@RFID macro. The DSM@RFID macro simply sets a local symbol for the DSMIREF macro, &*id, to the value of the REFID attribute.
3. If the id is over seven characters long, issues a message and the id is truncated to seven characters.
4. Tests the &I1@&*id symbol to determine if the id has been encountered before. If it has been, the following processing is performed:
   a. Sets the &@ilevel symbol to the sum of the existence of the three id symbols—I1@&*id, I2@&*id and I3@&*id. This equates to setting it to 3, if all three exist, 2 if only the first two exist and one if only the first one exists. This symbol is used in the attribute processing macros to determine what level of index entry tag is being processed.
   b. Puts the index terms themselves, which are in I1@&*id, and so on, into the &*t1, &*t2, and &*t3 symbols.
   c. Processes PG, PGREF, or PAGEREF using the DSM@PGRF macro. This may result in &*t4, the fourth index term, being redefined to something other than the page number symbol. It may also cause the &*x symbol to be set to "start," "order" or "end."

d. For level 1 and 2 index terms only, processes the SEE and SEEID attributes with the DSM@SEE and DSM@SEEI macro, respectively. This may set the &*r local symbol to the text of the index reference.

e. Tests the existence of the &*r local symbol which is set to the reference text if either the DSM@SEE macro or the DSM@SEEI macro has been used. If it does exist, &*x is set to "ref" because we will be generating an index reference rather than a real index entry. The index reference text is then moved into one of the local symbols &*t2 or &*t3 depending on the level of the index term being generated by the tag.[34]

f. Generates the index entry. The index terms are in &*t1, &*t2, &*t3 and &*t4. An additional .PI [Put Index] control word parameter may be in &*x. The hexadecimal "01"s are the delimiters on the control word line.

g. Ends the macro if this is not the first pass (&$PASS) or cross referencing (&SYSVARX) is not in effect.

h. If cross referencing is in effect, adds the current page number to the &IX@&*id array.

5. Tests the existence of the &I1@&*id symbol. If &I1@&*id doesn't exist, it indicates that the id being referenced has not been defined yet. This means we are dealing with a forward reference to an index entry. If we aren't cross referencing (&SYSVARX is not yes) or if this is the first pass, we don't need to do anything about saving the cross reference information because either we don't need it or we save it on the second pass.

   If we do need to save the cross reference information, the processing discussed below is performed. See "Cross-References" on page 147 for additional information on how cross referencing works.

   a. If IF@&*id doesn't exist (meaning we haven't encountered this id before at all) the ID is put into the cross reference array (&@xref@i). The &@xref@i array will be used to produce the cross reference listing for index ids.

   b. The element number from the cross reference array (&@xref@i) is remembered in IL@&*id so it can be replaced when we know what file the id is in.

   c. The page number is put into the next element of &IX@&*id.

   d. IF@&*id is set to "?." This array is supposed to contain the name of the file where the id was defined. If the id is never defined the file name will print out as "?" in the cross reference listing.

   e. If &IF@&*id already exists, the current page number is put into the array &IX@&*id.

DSMIDMMY

The DSMIDMMY APF processes the :I1-:I3 tags when an index is not being produced It simply consumes the residual text so it will not get printed.

DSMIEH

The DSMIEH macro processes the header information generated by the .IX [Index] control word. This macro takes over the function provided by the internally generated .IE [Index Entry] H control word. A macro named "IEH" is defined in the DSMINDEX macro and calls the DSMIEH macro to process the index heading parameter.

The index headers are produced as follows:

[34] A 1 is added to the level of the index term (&@ilevel) to determine which symbol should be set.

1. Tests the first parameter passed to this macro. If it is before "a" in the sorting sequence, it means that there are index entries beginning with special characters. In that case, the macro ends immediate because this section of the index gets no header.
2. Skips p22 (22 points).
3. If &$ENV indicates that there is a keep in progress, ends the keep. This is done so that we can start a keep without being in danger of getting an error message from SCRIPT/VS that a keep is already in progress.
4. Turns a keep on for 1.2 inches
5. Begins the "ieh" font for page printers or highlight font 2 for line devices. The alternate highlight font is also specified on the .BF [Begin Font] control word just in case neither of the other fonts can be started.
6. Indents 5.
7. Calculates the ending position of the box to be drawn as 7 plus the length of the heading to be generated.
8. Draws a box at position 4 and the calculated ending position provided that SYSOUT is not "PAGE," indicating that we are formatting for a page printer.
9. For page printers, sets indention back zero because the style of index headers is different for these devices.
10. Puts out the index heading using literal mode because we can't be sure what's in it.
11. Ends the box, if we had started one.
12. Skips 1 line.
13. Restores the previous font.
14. Restores indention to zero.

Index Attribute and Support Macros

DSM@IDS

See "Cross-References" on page 147 for the detailed discussion of the DSM@IDS macro. DSM@IDS processes all of the ID attributes.

DSM@IPRT

The DSM@IPRT macro processes the PRINT attribute of the :IH1-3 tags. This attribute specifies the text that is to be printed for the index term.

DSM@IPRT performs the following processing:

1. Saves the text string given (to be printed in the index) in &#it&@ilevel, where &@ilevel is 1,2, or 3 depending on the level of the index tag being processed.
2. Constructs a .PI [Put Index] control word sort key parameter in the caller's local symbol, &*k. (See "Special Techniques" on page 13 for details on the technique of setting the caller's local symbol when exiting a macro.) The hexadecimal "01"s are delimiters and the index term is placed in the first, second or third position based on the level of the index tag. If the hexadecimal "01"s were slashes, the &*k symbol would be set as follows:

```
    .se *k 'key /term///
  or
    .se *k 'key //term//
  or
    .se *k 'key ///term/
```

DSM@PGRF

The DSM@PGRF macro processes the PAGEREF attribute of the :I1-:I3 tags. It may be used to define a fourth index term, in which case the macro sets the caller's local symbol &*t. This attribute may also be used to specify additional index parameter information such as page ranges. In this case, the macro sets the caller's local symbol &*x.

See "Special Techniques" on page 13 for discussions on both the technique of examining the parameter passed in and setting the caller's local symbol.

DSM@PGRF performs the following functions:

1. Examines the parameter passed in for "start," "begin," "major," or "end."
2. If it is not any of the specific values tested for, sets the caller's local symbol &*t4 to the parameter sent in. This means that the user has explicitly set the 4th level index term and it will be used "as is."
3. If the parameter is "start" or "begin," sets the caller's local symbol &*x to "start." This means that a range of pages is being specified.
4. If it is "major," sets the caller's local symbol &*x to "order."
5. If it is "end," sets the caller's local symbol &*x to "end."

DSM@RIDI

The DSM@RIDI macro processes the REFID attribute of the :I2 and :I3 tags and provides the level one and level two index terms for creating the index entries. It performs the following processing:

1. Checks the length of the id parameter given. If the id is too long, it is truncated to 7 characters and a message is issued. Otherwise, the id's value is put into the &*id symbol.
2. Checks to see if the id (&I1@&*id) already exists. If so,
   a. Moves the saved index terms from &I1@&*id and &I2@&*id into the symbols &#it1 and &#it2. The &I1@&*id and &I2@&*id symbols are set when the ID attribute of the :I1 and :I2 tags respectively. The &#it1 and &#it2 symbols will be used to create the actual index entry control word.
   b. If cross referencing is in effect (&SYSVARX is "yes") and this is the first pass (&$PASS), puts the current page number into the symbol array &IX@&*id. This symbol is used only for the purposes of generating the cross reference listing at the end of the document.
3. Tests the existence of the level 1 id (&I1@&*id). If it doesn't exist it means we are dealing with a forward reference to an index entry. If we aren't cross referencing (&SYSVARX is not yes) or this is the first pass, we don't need to do anything about saving the cross reference information because either we don't need it or we will save it on the second pass.

   If we do need to save the cross reference information, the following processing is performed.

   a. If &IF@&*id doesn't exist, meaning we haven't encountered this id before at all, the id is put into the cross reference array (&@xref@i).

    1) The element number from the cross reference array (&@xref@i) is remembered in &IL@&*id so that the cross reference entry can be replaced when we know what file the id is in.
    2) IF@&*id is set to "?." This symbol is supposed to contain the name of the file where the id was defined. If the id is never defined the file name will print out as "?" in the cross reference listing.
    3) The page number is put into &IX@&*id.

  b. If &IF@&*id already exists, adds the current page number to the array &IX@&*id.

DSM@SEE

The DSM@SEE macro processes the SEE attribute on the index tags. (:I1-3 and :IREF).

Stacks a control word to be processed in the calling macro and assigns the attribute value to the caller's local symbol &*r. This technique is discussed in detail in "Special Techniques" on page 13.

DSM@SEEI

The DSM@SEEI macro processes the SEEID attribute of the index tags. The index reference text is returned in the caller's local symbol &*r. This technique is discussed in detail in "Special Techniques" on page 13. DSM@SEEI performs the following functions:

1. If the id name is over seven characters, it issues a message and and truncates it to seven characters.

2. If the id isn't over seven characters, saves the id in a local symbol, *id.

3. If I1@&*id exists (meaning that the id has been encountered before), the following processing is performed.

   a. The text of the index reference is put into a local symbol, &*r.

   b. The id may have been defined at any of the 3 levels of indexing. The symbol &*r is constructed of the first and/or second and/or third entries depending on what symbols exist. For example, let's assume the id referenced on the SEEID attribute was defined on an :I3 tag for "rebates." Let's assume further that the the first and second level terms were "housing" and "costs." The &*r symbol would end up being set here to "housing, costs, rebates."

   c. If cross referencing is in effect or this is the first pass (&$PASS is 1 and &SYSVARX is "yes") the current page number is added to &IX@&*id.

4. If the level 1 id (&I1@&*id) doesn't exist it means we are dealing with a forward reference to an index entry. If we aren't cross referencing (&SYSVARX is not yes) or this is the first pass, we don't need to do anything about saving the cross reference information. We will save it on the second pass or we don't need it.

   If we do need to save the cross reference information, the following processing is performed:

   a. If &IF@&*id doesn't exist, meaning we haven't encountered this id before at all, different actions need to be taken:

      1) An entry is made in the cross reference array (&@xref@i) for the id.
      2) The element number from the cross reference array (&@xref@i) is remembered in &IL@&*id so it can be replaced when we know what file the id is in.
      3) &IF@&*id is set to "?." This symbol is supposed to contain the name of the file where the id was defined. If the id is never defined the file name will print out as "?" in the cross reference listing.

   b. The current page number is added to the array &IX@&*id.

5. The caller's local variable &*r is set to "?." This technique is discussed in "Special Techniques" on page 13.

Modifications to Index Tags

It is difficult to think of any modifications that can be made to the starter set index tags. However, there are several aspects of the native SCRIPT/VS index functions which can be altered. The primary one is to change the words "See" and "See also" to something else—for example, translating them into another language. This change is possible, however, it requires changing the DSMCSPFR module of SCRIPT/VS. This module contains all of the literal constants that are built into SCRIPT/VS, just as the DSM#SETS and DSM#MSG macros contain all of the literals built into the starter set. See the chapter on tailoring SCRIPT/VS to your installation in the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for details on how to change DSMCSPFR.

There are two other modifications which are frequently requested. The first is the ability to alter the sorting order for the index. The second is the ability to change the format of the index entries such that the page numbers are lined up on the right hand side of the column. Unfortunately, neither of these functions can be altered within SCRIPT/VS.

Cross-References

Overview

The starter set provides the capability to cross reference headings, figures, list items, footnotes, and index entries. The cross reference processing for the first four types of text elements is described here. Index entry cross referencing is described in "Indexing" on page 137.

Cross reference processing occurs at different times in the starter set and at different levels. There are four main types of activity:

- Processing ID attributes
- Resolving cross references within the document
- Printing a cross reference listing
- Creating and using a file of id information.

These activities are highly interrelated. The cross reference material printed at the end of the document is contained in several arrays and variables. These arrays and variables are created by the macro that processes the ID attributes. These same variables are used to resolve the cross references and produce the SYSVAR "W" id file at the end of the formatting run.

For example, cross referencing for a heading involves the following processing steps:

1. When an :H1 tag is encountered with an ID attribute on it, several processing steps are performed:

   a. The DSMHEAD1 APF, which processes the heading, calls the DSM@IDS macro.

   b. The DSM@IDS macro performs several processing functions:

      1) The id, the text associated with it, and the page reference are saved in a set of variables which will be used to resolve any references to this heading.
      2) The id, the file name, and the page reference are added to the array which produces the cross reference listing.
      3) The id, the text of the reference, and the page reference are added to the array which will be used to produce the SYSVAR "W" id file.

   See Figure 21 on page 148 for a picture of the relationships between these arrays and macros. The names of the arrays that are built are shown on the right side of the figure.

2. When a :HDREF tag is encountered:

   a. The variables set up by the DSM@IDS macro are used to resolve the cross reference and insert the appropriate text into the document where the reference was made.

   b. If the ID information has not been defined, but a SYSVAR "R" file of ids has been used, the id information from that file will be used to resolve the cross reference.

```
*------------*                                *---------*
|   HO - 6   |--*              *--> BUILDS -->| @xref@h |
|if ID given |  |              |              | @writ@h |
*------------*  |              |              *---------*
                |              |
*------------*  |              |              *---------*
|    FIG     |--|              |--> BUILDS -->| @xref@f |
|if ID given |  |  *----------*|              | @writ@f |
*------------*  |->| DSM@IDS  ||              *---------*
                |  *----------*|
*------------*  |              |              *---------*
|     FN     |--|              |--> BUILDS -->| @xref@n |
|if ID given |  |              |              | @writ@n |
*------------*  |              |              *---------*
                |              |
*------------*  |              |              *---------*
|  I(H)1-3   |--|              |--> BUILDS -->| @xref@i |
|if ID given |  |              |              | @writ@i |
*------------*  |              |              *---------*
                |              |
*------------*  |              |              *---------*
|  LI        |--*              *--> BUILDS -->| @xref@d |
|if ID given |                                | @writ@d |
*------------*                                *---------*
```

Figure 21. Processing ID Attributes: All of the ID attributes in the starter set are processed by the DSM@IDS macro. It builds five different symbol arrays which contain the necessary information to produce the cross reference listing for the five types of ids.

   c. If the id cannot be found in either place, a standard message is used instead for the cross reference.

   d. The page number where the reference was made is added to the array of page numbers to be used in the cross reference listing.

3. At the end of the document, the DSM#XLST macro will be called either by the :EGDOC tag or by the epifile in DSMPROF3. This macro initiates the following processing:

   a. Activates the &@xref@h array that has been built dynamically by DSM@IDS as ID attributes were found. Each line of this array contains a call to the DSM#XRFH macro.

      1) The first time DSM#XRFH is called, it calls DSM#SETX which formats the appropriate heading in the cross reference section.

      2) The DSM#XRFH macro uses the variables set up by DSM@IDS to print the first part of the cross reference line which includes the id, the file name, and the original page number.

      3) It uses the page number array to print the rest of the entry.

   Figure 22 on page 149 shows the processing steps involved in printing the cross reference. This will occur only if SYSVAR "X" was not set to "no" on the SCRIPT command.

4. Also at the end of the document, if SYSVAR "W" has been specified, the processing necessary to produce the id file is performed.

```
*--------*
|DSMEGDOC|-*
*--------* |  *--------*    *-------*  *--------*      *---------*
           |->|DSM#XLST|--->|@xref@h|->|DSM#XRFH|---->|DSM#SETX |
*--------* |  *--------*  | *-------*  *--------*  |   *---------*
|DSMPROF3|-*              | *-------*  *--------*  |
*--------*                |->|@xref@h|->|DSM#XRFF|--|
                          | *-------*  *--------*  |
                          | *-------*  *--------*  |
                          |->|@xref@h|->|DSM#XRFN|--|
                          | *-------*  *--------*  |
                          | *-------*  *--------*  |
                          |->|@xref@h|->|DSM#XRFI|--|
                          | *-------*  *--------*  |
                          | *-------*  *--------*  |
                          *->|@xref@h|--|DSM#XRFD|--*
                            *-------*  *--------*
```

Figure 22. Macros and Symbol Arrays Used to Produce the Cross Reference Listing: The &@xref@h, &@xref@f, &@xref@n, &@xref@i and &@xref@d symbol arrays contain one line for each id used in the document. The DSM#XRFF, DSM#XRFH, DSM#XRFI, DSM#XRFD, and DSM#XRFN macros produce the actual listing and the DSM#SETX macro produces the section headings within the listing and is called only once for each set of ids.

a. The DSM#WRIT macro is invoked which in turn labels the file and processes the &@writ@d, &@writ@f, &@writ@h, and &@writ@n arrays which call the DSM#WRTD, DSM#WRTF, DSM#WRTH, and DSM#WRTN macros.

b. These arrays were created dynamically by the DSM@IDS macro when the ids were processed. These macros write .SE [Set Symbol] control word lines out to the SYSVAR "W" file.

Initialization for Cross Referencing

During initialization of the starter set in DSMPROF3 and the macros it calls, several functions are performed relevant to cross referencing. These are:

1. The DSM#SET macro pre-defines the cross reference arrays (&@xref@d, &@xref@f, &@xref@h, &@xref@i and &@xref@n) to be comments. This is done to prevent them from being undefined symbols when we print out the cross reference listing. These arrays are expected to contain information regarding the ids in the document. If there were, for example, no index ids used in the document, the &@xref@i array wouldn't have anything in it and it would print out as a undefined symbol if it weren't pre-defined.

2. The symbol arrays used to write out the SYSVAR "W" file are also initialized by the DSM#SET macro starter set for the same reason.

3. The value of &SYSVARX is established as either "yes" or "no" by the DSM#SETV macro.

4. The SYSVAR "R" file, if one was specified on the command line, is imbedded by DSM#SETV to define symbol values to be used to resolve cross references that would be otherwise unresolvable.

5. DSMPROF3 maps the cross reference tags to their appropriate macros.

Processing ID Attributes

All of the ID attributes in the starter set are processed by the DSM@IDS macro. It does three things:

1. Saves the ID information for use in cross referencing.
2. Creates the necessary arrays to produce the cross reference listing.
3. Creates the necessary arrays to produce the SYSVAR "W" file.

DSM@IDS

The DSM@IDS macro is used to process the ID attributes for the heading tags (:H0-H6), the list item tag (:LI), the footnote tag (:FN), the figure tag (:FIG) and the index tags (:I1-3 and :IH1-3).

Its primary function is to create two symbols that contain the text of the reference and the page on which it appears. These are used for resolving references to ids.

The DSM@IDS macro performs the following processing:

1. The macro ends immediately if this is not the first pass (&$PASS) because ID information is collected only on the first pass.
2. The type of id being processed is indicated in the &@tg symbol. Its value may be "n" for footnote, "h" for heading, "d" for list item, "i" for index, and "f" for figure. This symbol is needed in both uppercase and lowercase format so it is initially set in lowercase by the calling macro and then folded to uppercase here in &@TG.
3. The id name itself is in the &* symbol. Its length is checked to make sure it is not over 7 characters long and it is put into a local symbol, &*id. If the length of the id name is more than 7 characters, it is truncated to 7 characters and a message is issued.
4. The existence of the id is tested by checking the symbol &@TG.1@&*id which for a heading id of "intro" would resolve like this:

   ```
   &@TG.1@&*id
   &@TG.1@intro
   &H1@intro
   ```

   This symbol will exist only if the id has already been defined by an ID attribute on a heading tag. A message is issued if the id already exists (that is, it is a duplicate.) The macro ends and this definition of the id is ignored.
5. If cross referencing was requested (&SYSVARX is "yes") or we are going to produce a SYSVAR "W" file, the next thing to do is create the appropriate cross reference array entries for these. If we are not cross referencing or creating the ID file, we have fewer things we need to remember about the ids.

   a. The cross reference arrays are named &@xref@&@tg where &@tg resolves to d,f,h,i or n depending on the type of id being processed. An entry is made in the appropriate array. It contains a call to the appropriate macro (DSM#XRFD, DSM#XRFF, DSM#XRFH, DSM#XRFI or DSM#XRFN) with the id value as a parameter. This will cause the id information to appear in the cross reference listing.

      For example, for a heading ID with a value of "intro" the following line is put into the &@xref@h array:

```
.se @xref@&@tg.() '.dsm#xrf&@tg. &*id
.se @xref@&@tg.() '.dsm#xrf&@tg. intro
.se @xref@&@tg.() '.dsm#xrfh intro
.se @xref@h() '.dsm#xrfh intro
```

b. The id file arrays are named &@writ@&@tg where &@tg resolves to d,f,h,i or f depending on the type of id. An entry is made in the appropriate array. It contains a call to the appropriate macro (DSM#WRTD, DSM#WRTF, DSM#WRTH, DSM#WRTI or DSM#WRTN) with the id value as a parameter. This will cause the id information to be written out to the SYSVAR "W" file at the end of the formatting run.

   Note: The id information for the index entries is written out to the &@writ@i array. However, these are never processed because index id information is not saved in the SYSVAR "W" file. This is done because it is easier to write out the information than to not write it out, given the way the DSM@IDS macro processes the ids.

c. The next step is to define some special symbols whose names are dependent on the id name (&*id) and the type of id (&@TG).

   1) The &@TG.F@&*id will become &HF@intro for a heading id named "intro." It will contain the name of the file in which the id was defined. If this symbol already exists it means that the id has been referenced before but has not been defined until now. If this is the case, an incomplete entry was made in the &@xref@&@tg array and it must now be replaced. Correcting it involves undefining the previous incomplete entry whose element number was saved at the time in &@TG.L@&*id. This element is set to "off" because the correct element has just been defined above. It is necessary to save element 0 of the array before resetting the element to "off" and then restoring it in order to prevent it from being incorrect afterwards.

   2) Then the file name (&$FNAM) is saved in the &TG.F@&*id symbol.

6. The page number is saved in a symbol whose name depends on the id type (&@TG) and the id name (*id).

7. The next step is to save the text associated with the id. This is the text of the heading, the figure number, the list item id, the footnote number, or the index term. The id name is part of the symbol name to make it unique.

   For index id's this is a little more complicated. The &@ilevel symbol contains either a "1," a "2" or a "3" depending on the level of the index entry for which an id is being processed. We need to remember all of the terms (up to 3) associated with this id. For third level entries, we set a symbol for the second and first terms and for the current term. For second level entries, we set a symbol for the first level and for the current term. For first level entries, we set a symbol for the current term.

Processing Cross Reference Tags

DSMHDREF

The DSMHDREF APF processes references made to heading ids using the :HDREF tag. It functions as follows:

1. Processes the REFID attribute using the DSM@RFID macro to get the id into the local symbol &*id.

2. Processes the PAGE attribute using the DSM#YESN macro which will set the &*yesno symbol to "no" or "yes" depending on the value of the PAGE attribute.

3. Truncates the id name to 7 characters if it is longer than that and issues a message.

4. Determines if the real id information, generated during this formatting run, is available or if the information read in from a SYSVAR "R" id file will have to be used. The only difference between the symbols used is in whether or not the first letter of the symbol name is a capital H. Capital H denotes real id information saved during this run for headings. Lowercase h names are information set by a SYSVAR "R" file. The SYSVAR "R" file, if any, was imbedded during initialization by the DSM#SETV macro.

   We assume first that we'll use real information. Then we test if the &H1@&*id or &h1@&*id symbols exist. If neither one exists, we skip to the "unknown" label because we have no information available to use. If the "real" symbol, &H1@&*id, doesn't exist we reset &*H to a lowercase "h" to use the id information set by the SYSVAR "R" file.

5. If the heading id exists:

   a. Turn substitution off.

   b. Save the value of the control word separator in a local symbol and then turn the control word separator off. This is done because we don't want SCRIPT/VS to get confused by any semi-colons which happen to be in the heading text.

   c. Save the page number (&) in the &*p local symbol.

   d. Selects the proper level of quotation marks to use around the heading. The "proper" level depends on the current level of quotation nesting. The quotation nesting level is in &@nest@q and the quotation marks are in the &@oquote and &@cquote symbols.

   e. Sets up the local symbol &*r to contain the page reference part of the cross reference but only if the PAGE attribute value was "yes" or the heading being referenced is not on the current page. The page number of the reference is in &HP@&*id and the current page number has been put into &*p. These two symbols are compared to determine if the heading occurred on the current page.

      The page reference is constructed of the &LL@onpge symbol (which is set to "on page" in the DSM#SETS macro) and the page number.

   f. Turn substitution back on.

   g. Insert the heading reference into the text. The first symbol, &*o, contains the opening quotation mark. The second symbol, &H1@&*id, contains the text of the heading. The third symbol, &*c, contains the closing quotation mark and the fourth symbol, &*r, may be null or may contain the page reference.

   h. Restores the control word separator.

   i. Saves the page number in the &HX@&*id symbol array. This is done only on the first pass and only if we are going to produce a cross reference listing (&SYSVARX is yes). The &HX@&*id array contains the page numbers of all the references to the heading. If the array already exists, it means there have been other references made to this same heading.

6. If the heading id information does not exist (unknown):

   a. Spelling verification is turned off.

   b. Inserts some standard text which includes the id value in place of the heading reference. The &LL@H symbol is set to "Heading" in DSM#SETS. The &LL@unkn symbol is set to "unknown" in DSM#SETS.

   c. Turn spelling verification back on.

   d. If cross referencing (&SYSVARX is "yes") is in effect and this is the first pass (&$PASS is 1):

1) The &HF@&*id symbol is tested to see if this id has been encountered yet. If it has, &HF@&*id will contain the name of the file in which it was defined. If it hasn't been defined yet, &HF@&*id will not exist, in which case:

   a) An entry is made in the cross reference array (&@xref@h) for the id. The &@xref@h array will be used to produce the cross reference listing. Each line of this array contains a call to the DSM#XRFH macro with the id as the parameter.
   b) HL@&*id is set to the element number just used in the &@xref@h array. This is done because eventually we'll encounter the id's definition and we'll want to be able to update the entry we made in the array. This updating will be done in the DSM@IDS macro when the ID is processed and will use the element number we saved here in the &HL@&*id symbol.
   c) The &HF@&*id symbol is supposed to contain the name of the file where the id was defined. For ID's that haven't been defined, we put a "?" in this symbol so that something will print out in the cross reference listing.

2) The page number is saved in the &HX@&*id symbol array. The &HX@&*id array contains the page numbers of all the references to the heading. If the array already exists, it means there have been other references made to this same heading.

DSMFGREF

The :FIGREF tag is processed by the DSMFGREF APF which inserts the appropriate reference text. This tags works only for those figures which have an ID attribute and which have been labelled with a figure number by the :FIGCAP tag. If the ID hasn't been defined, the "Figure Unknown" reference is used. If there was no :FIGCAP tag, the figure number in the reference will be wrong because the id information will point to what the figure number would have been if a figure number had been assigned to the figure.

The :FIGREF tag is processed in the following manner:

1. Processes the REFID attribute using the DSM@RFID macro to get the id into the local symbol &*id.

2. Processes the PAGE attribute using the DSM#YESN macro which will set the &*yesno symbol to "no" or "yes" depending on the value of the PAGE attribute.

3. Truncates the id name to 7 characters if it is longer than that and issues a message.

4. Determines is the real id information, generated during this formatting run, is available or if the information read in from the SYSVAR "R" id file will have to be used. The only difference between the symbols used is whether or not the first letter of the symbol name is a capital F. Capital F denotes real id information saved during this run. Lowercase f names represent information set by a SYSVAR "R" file.

   We assume first that we'll use the real information. Then we test if the &F1@&*id or &f1@&*id symbols exist. If neither one exists, we skip to the "unknown" label because we have no information available to use. If the "real" symbol, &F1@&*id, doesn't exist we reset &*F to a lowercase f to use the id information set by the SYSVAR "R" file.

5. Tests if the figure id exists. If so:

   a. Saves the page number in a local symbol, &*p.

   b. Sets the local symbol &*r to contain the page reference part of the cross reference but only if the PAGE attribute value was "yes" or the figure being referenced is not on the current page. The page number of the figure being referenced is in &FP@&*id and the current page number has been put into &*p. These two symbols are compared to determine if the figure occurred on the current page.

      The page reference is constructed of the &LL@onpge symbol, which is set to "on page" in the DSM#SETS macro, and the page number.

c. Inserts the figure reference into the text. The reference is constructed of the word "Figure" (&LL@F), a required blank, the figure number (&&*F.1@&*id) and a possible page reference (&*r).

d. Saves the page number in the &FX@&*id symbol array. This is done only on the first pass and only if we are going to produce a cross reference listing (&SYSVARX is "yes"). The &FX@&*id array contains the page numbers of all the references to the figure. If the array already exists, it means there have been other references made to this same figure.

6. If the figure id information does not exist (unknown):

   a. Spelling verification is turned off.

   b. Inserts some standard text which includes the id value in place of the figure reference. The &LL@F symbol is set to "Figure" in DSM#SETS. The &LL@unkn symbol is set to "unknown" is DSM#SETS.

   c. Spelling verification is turned back on.

   d. If cross referencing (&SYSVARX) is in effect and this is the first pass (&$PASS is 1):

      1) The &FF@&*id symbol is tested to see if this id has been encountered yet. If it has, &FF@&*id will contain the name of the file in which it was defined. If it hasn't been defined yet, &FF@&*id will not exist, in which case:

         a) An entry is made in the cross reference array (&@xref@f) for the id. The &@xref@f array will be used to produce the cross reference listing. Each line of this array contains a call to the DSM#XRFF macro with an id as a parameter.
         b) &FL@&*id is set to the element number just used in the &@xref@f array. This is done because eventually we'll encounter the id's definition and we'll want to be able to update the entry we made in the array. This updating will be done in the DSM@IDS macro when the ID is defined and will use the element number we saved here in the &FL@&*id symbol.
         c) The &FF@&*id symbol is supposed to contain the name of the file where the id was defined. For IDs that haven't been defined, we put a "?" in this symbol so that something will print out in the cross reference listing.
         d) The page number is saved in the &FX@&*id symbol array. The &FX@&*id array contains the page numbers of all the references to the figure. If the array already exists, it means there have been other references made to this same figure.

DSMLIREF

The DSMLIREF APF process the :LIREF tag. The list item reference consists of the list item identifier that is on the item. This means list item references are meaningful only for ordered list items where the identifier is unique.

DSMLIREF performs the following functions:

1. Processes the REFID attribute using the DSM@RFID macro to get the id into the local symbol &*id.

2. Processes the PAGE attribute using the DSM#YESN macro which will set the &*yesno symbol to "no" or "yes" depending on the value of the PAGE attribute.

3. Checks the length of the id name and truncates it to 7 characters if it is longer. A message is issued if the value is too long.

4. Determine if the real id information generated during this formatting run is available for use, or if the information read in from a SYSVAR "R" id file will have to be used. The only difference between the symbols used is in whether or not the first letter of the symbol name

is a capital D. Capital D denotes real id information saved during this run and lowercase d names are information set by a SYSVAR "R" file.

We assume first that we'll use real information. Then we test if the &D1@&*id or &d1@&*id symbols exist. If neither one exists, we skip to the "unknown" label because we have no information available to use. If the "real" symbol, &D1@&*id, doesn't exist we reset &*D to a lowercase d to use the id information set by the SYSVAR "R" file.

5. If the list item id exists DSMLIREF:

   a. Turns substitution off.

   b. Saves the control word separator in a local symbol and then turns the control word separator off.

   c. Saves the page number in a local symbol, &*p.

   d. Sets up the local symbol &*r to contain the page reference part of the cross reference but only if the PAGE attribute value was "yes" or the list item being referenced is not on the current page. The page number of the reference is in &DP@&*id and the current page number has been put into &*p. These two symbols are compared to determine if the list item occurred on the current page.

      The page reference is constructed of the &LL@onpge symbol (which is set to "on page" in the DSM#SETS macro) and the page number.

   e. Turns substitution back on.

   f. Inserts the list item reference into the text. The reference is constructed of the list item identifier (&D1@&*id) and a possible page reference (&*r).

   g. Restores the control word separator.

   h. Saves the page number in the &DX@&*id symbol array. This is done only on the first pass and only if we are going to produce a cross reference listing (&SYSVARX is "yes"). The &DX@&*id array contains the page numbers of all the references to the list item. If the array already exists, it means there have been other references made to this same list item.

6. If the list item id information does not exist (unknown) DSMLIREF:

   a. Spelling verification is turned off.

   b. Inserts some standard text (which includes the id value) in place of the figure reference. The &LL@LI symbol is set to "LI" in DSM#SETS.

   c. Turns spelling verification back on.

   d. If cross referencing (&SYSVARX is "yes") is in effect and this is the first pass (&$PASS is 1) DSMLIREF:

      1) Tests the &DF@&*id symbol to determine if this id has been encountered yet. If it has, &DF@&*id will contain the name of the file in which it was defined. If it hasn't been defined yet, &DF@&*id will not exist, in which case:

         a) Adds an entry to the cross reference array (&@xref@d) for the id. The &@xref@d array will be used to produce the cross reference listing. Each line of this array contains a call to the DSM#XRFD macro with an id as a parameter.
         b) Sets &DL@&*id to the element number just used in the &@xref@d array. This is done because eventually we'll encounter the id's definition and we'll want to be able to update the entry we made in the array. This updating will be done in the DSM@IDS macro when the ID is processed and will use the element number we saved here in the &DL@&*id symbol.

c) Sets &DF&*id. The &DF@&*id symbol is supposed to contain the name of the file where the id was defined. For ID's that haven't been defined, we put a "?" in this symbol so that something will print out in the cross reference listing.

d) Saves the page number in the &DX@&*id symbol array. The &DX@&*id array contains the page numbers of all the references to the list item. If the array already exists, it means there have been other references made to this same list item.

DSMFNREF

The :FNREF tag is processed by the DSMFNREF APF. Processing footnote references is simpler than processing other kinds of references. The reference itself is just the footnote number.

:FNREF is processed as follows:

1. Processes the REFID attribute using the DSM@RFID macro to get the id into the local symbol &*id.

2. If the length of the id is longer than 7 characters, truncates it and issues a message.

3. Checks the existence of the id (&N1&*id). If it already exists, it means that the footnote which defines the id has already been encountered. In that case we call DSM#SUPR with the footnote number (&N1@&*id) to produce the superscript footnote call-out.

4. Puts out a footnote call-out of "00" if the footnote number (&N1@&*id) doesn't exist and the id number from the SYSVAR "R" id file (&n1@&*id) doesn't exist. This is done by calling the DSM#SUPR macro with "00" as the parameter.

5. If the SYSVAR "R" id value exists, calls DSM#SUPR with its value instead of &N1&*id.

6. Collects the cross reference listing information only on the first pass and only if &SYSVARX is "yes." If the id information exists (&N1@&*id) we save the page number in the &NX@&*id symbol. This array will contain an entry for each time the footnote was referenced.

7. If the id information doesn't exist, meaning that the footnote being referenced hasn't been encountered yet, saves more cross reference information. This is because we need to be able to correct the cross reference listing entries when the footnote is encountered.

   If the file name isn't known (&NF@&*id) then we make an entry in the &@xref@n array. The entry is a call to the DSM#XRFN macro with the id as a parameter. The &@xref@n array is used to actually create the cross reference listing. The array element number is saved in &NL@&*id so that we can replace it with the correct information when we know where the footnote is actually defined.

   The file name is supposed to be in &NF@&*id. This symbol is set to a question mark since we don't know where the footnote is yet. The page number is then saved in the &NX@&*id array.

DSM@RFID

DSM@RFID processes the REFID attributes of the :HDREF, :LIREF, :FIGREF, and :FNREF tags.

All this macro does is to put the attribute value passed in &* into the caller's local symbol named &*id by stacking a line on the .ME [Macro Exit] control word line. This technique, of setting the caller's local symbol, is described in detail in "Special Techniques" on page 13.

DSM#YESN

DSM#YESN processes various attributes which have yes/no values; specifically the PAGE attribute of the :HDREF, :FIGREF, and :LIREF tags.

The DSM#YESN macro performs the following processing:

1. Searches the attribute value for yes or no. This technique is described in detail in "Special Techniques" on page 13.
2. Returns an answer of either "yes" or "no" in the caller's local symbol, &*yesno. This technique is explained in "Special Techniques" on page 13.
3. Sets &*yesno to "yes" if the attribute value was something other than "yes" or"no."

Cross Reference Listing Macros

DSM#XLST

The DSM#XLST macro produces the cross reference listing. This macro is called only if &SYSVARX is "yes" on the last pass. It may be called either from the epifile or from the :EGDOC tag APF. The listing is produced as follows:

1. The first thing this macro does is redefine its first line to exit immediately. This is done to prevent the cross referencing listing from being produced by both the :EGDOC tag and the epifile. This technique is discussed in "Special Techniques" on page 13.
2. Any open lists, figures, quotations, and so on are ended by calling the DSM#RSET macro.
3. The DSM#DUPL macro is called to get to the next/odd page.
4. Running headings and footings are suppressed because we don't particularly want them on the cross reference listing pages.
5. The control word separator is reset to its default value (;).
6. The column layout is redefined to be a single column starting at the left hand margin.
7. Column length (&$CL) is reset to its default.
8. Formatting is set to "left" because we don't want lines to be horizontally justified in the listing.
9. The body font (hi0) is restarted just to make sure that's what font we are in.
10. Spelling verification is turned off because there's no need to check spelling here and it would be inefficient.
11. Each of the five sections of the listing is formatted as follows:
    a. If the array which contains the listing information doesn't exist there is nothing to do and we go on to the next section. The arrays are &@xref@d, &@xref@f, &@xref@h, &@xref@i and &@xref@n.
    b. The array separator is set to the control word separator. This is done by setting the separator to be the three characters necessary to create a symbol name. The &@ symbol is then defined to be a semi-colon.
    c. The arrays are then called. Each line of the array contains a call to another macro with an id as a parameter. The macros called are named DSM#XRFD, DSM#XRFF, DSM#XRFH, DSM#XRFI and DSM#XRFN.

12. This macro also produces the imbed trace if there is something in the &@imtrace symbol.

    a. Resets indention and offset to 0.

    b. Skips 4 lines.

    c. Performs a page eject if there isn't 3 inches left on the page.

    d. Begins highlight font 2 (hi2).

    e. Formats the heading which consists of a box from left to right with the words "Imbed Trace" centered inside it. The words are contained in the &LL@ImTrc symbol which is defined in the DSM#SETS macro during initialization.

    f. Spaces 2 lines.

    g. Restores the previous font.

    h. Turns off formatting and the overdraw option is changed to EXTEND.

    i. Defines a tab at 12 ems.

    j. Defines the array separator to cause a break. This is done by setting the array separator to be the characters necessary to create a symbol name and then setting the symbol (&@) to be a .BR [Break] control word.

    k. The &@imtrace array is then called. This array was defined by the DSMIM macro with information regarding each imbed being an element in the array.

DSM#SETX

The DSM#SETX macro is used to create the cross reference listing header only. This macro is called once each by DSM#XRFD, DSM#XRFF, DSM#XRFH, DSM#XRFI, and DSM#XRFN. Its purpose is to generate the heading for the section, which it does in the following way:

1. Skips 4 lines.

2. Resets left and right indention to the zero.

3. Turns a keep on for 2 inches.

4. Sets the offset for the second and subsequent lines of the listing for each id to 32m. This value is modified for the index cross reference listing because the format for these is slightly different.

5. Establishes the tab positions for all devices in em-spaces.

6. Redefines the array separator to be a comma and a blank. The page numbers on which each id has been referenced have been saved in an array. When they are printed out, they will be separated by a comma and a blank.

7. Begin highlight font 2.

8. Starts a box for the full length of the line.

9. Centers the section heading. The parameter passed into this macro will be an D, F, H, I or N which when combined with &LL@. will produce a symbol which resolves to "Heading," "List Item," "Figure," "Footnote" or "Index," respectively. These symbols are defined in the DSM#SETS macro.

10. Ends the box.

11. Spaces 2 lines.

12. Prints the column headings using predefined literal symbols which are defined in the DSM#SETS macro during initialization. &LL@File is set to "File." &LL@Page is set to "Page." &LL@&*1, again, will resolve to "Heading," "Figure," and so on depending on the parameter passed in. &LL@Refs is set to "References."
13. Restores the previous font.
14. Spaces 1 line.

DSM#XRFF

The DSM#XRFF macro formats the cross reference entry for a figure "id." The cross reference listing is produced by the DSM#XLST macro which issues the &@xref@f array. This array contains a call to the DSM#XRFF macro for each figure id that has been defined or used.

DSM#XRFF performs the following processing:

1. If the id being formatted was referred to but never defined, &F1@&*id and &FP@&*id will not exist and "?" will be printed instead of the text and the page number.
2. If it doesn't exist the page number array &FX@&*id is set to null.
3. The first time through, it calls DSM#SETX to create the listing header. This technique of performing some processing only the first time is explained in detail in "Special Techniques" on page 13.
4. The cross reference entry is printed with an offset for potential wrap-around of page numbers onto a second line. Tabs are used to move from column to column. The tabs were defined in the DSM#XLST macro. The file name is in &FF@&*id. The text is in &F1@&*id and the page numbers are in &FX@&*id.

DSM#XRFH

The DSM#XRFH macro formats the cross reference entry for a heading "id." The cross reference listing is produced by the DSM#XLST macro which issues the &@xref@h array. This array contains a call to the DSM#XRFH macro for each heading id that has been defined or used.

DSM#XRFH performs the following processing:

1. If the id being formatted was referred to but never defined, &H1@&*id and &HP@&*id will not exist and "?" will be printed instead of the text and the page number.
2. The first time through, it calls DSM#SETX to create the listing header. This technique of performing some processing only the first time is explained in detail in "Special Techniques" on page 13.
3. The cross reference entry is printed with an offset for potential wrap-around of page numbers onto a second line. Tabs are used to move from column to column. The tabs were defined in the DSM#XLST macro. The file name is in &HF@&*id and the page the heading was formatted on is in &HP@&*id. The text of the heading is in &H1@&*id.
4. The &HX@&*id symbol contains the pages on which the heading was references. If &HX@&*id contains any entries, they are printed out.

DSM#XRFN

The DSM#XRFN macro formats the cross reference entry for a footnote "id." The cross reference listing is produced by the DSM#XLST macro which issues the &@xref@n array. This array contains a call to the DSM#XRFN macro for each footnote id that has been defined or used.

DSM#XRFN performs the following processing:

1. If the id being formatted was referred to but never defined, &N1@&*id and &NP@&*id will not exist and "?" will be printed instead of the text and the page number.

2. The page number array (&NX@&*id) is set to null if it doesn't exist.

3. The first time through, it calls DSM#SETX to create the listing header. This technique of performing some processing only the first time is explained in detail in "Special Techniques" on page 13.

4. The cross reference entry is printed with an offset for potential wrap-around of page numbers onto a second line. Tabs are used to move from column to column. The tabs were defined in the DSM#XLST macro. The file name is in &NF@&*id. The text is in &N1@&*id and the page numbers are in &NX@&*id.

DSM#XRFD

The DSM#XRFD macro formats the cross reference entry for a list item "id." The cross reference listing is produced by the DSM#XLST macro which issues the &@xref@d array. This array contains a call to the DSM#XRFD macro for each index id that has been defined or used.

DSM#XRFD performs the following processing:

1. If the id being formatted was referred to but never defined, &D1@&*id and &DP@&*id will not exist and "?" will be printed instead of the text and the page number.

2. The page number array &DX@&*id is set to null if it doesn't exist.

3. The first time through it calls DSM#SETX to create the listing header. This technique of performing some processing only the first time is explained in detail in "Special Techniques" on page 13.

4. The cross reference entry is printed with an offset for potential wrap-around of page numbers. Tabs are used to move from column to column. The tabs were defined in the DSM#XLST macro.

   The file name is in &DF@&*id. The text is in &D1@&*id and the page numbers are in &DX@&*id.

DSM#XRFI

The DSM#XRFI macro formats the cross reference entry for an index "id." The cross reference listing is produced by the DSM#XLST macro which issues the &@xref@i array which contains a call to the DSM#XRFI macro for each index id that has been defined or used.

DSM#XRFI performs the following processing:

1. The first time through it calls DSM#SETX to create the listing header. This technique of performing some processing only the first time is explained in detail in "Special Techniques" on page 13.

2. If the id being formatted was referenced but never defined, &I1@&*id and &IP@&*id will not exist and "?" will be printed instead of the text and the page number.

3. The cross reference entry is printed with an offset for potential wrap-around of page numbers. The offset, &@xref@of is set in DSM#SETX. In the case of index id's the offset is reduced a little because there's so much more information to be listed.

   Tabs are used to move from column to column. The tabs were defined in the DSM#XLST macro. The file name is in &IF@&*id. For each index id there may one, two or three levels of information to be listed depending on the level at which the id was defined. The existence of the information is tested before it is formatted. The text is in &I1@&*id, &I2@&*id, &I3@&*id and the page numbers are in &IX@&*id.

Producing the SYSVAR 'W' Id File

The id information which will be written out to the SYSVAR "W" file is collected by the DSM@IDS macro when it processes the ID attributes. The reference text and the page number are both saved. The lines written out are actually .SE [Set Symbol] control word lines which define symbols to contain the text and page number. When this file is read in as a SYSVAR "R" file, the symbols are set and may be used to resolve forward cross references.

DSM#WRIT

The SYSVAR "W" file of id information is produced by the DSM#WRIT macro. This macro is called either by the epifile or by the APF which processes the :EGDOC tag, DSMEGDOC. The file is created with the following processing:

1. The first thing this macro does is redefine its first line to exit immediately. This is done to prevent the cross referencing listing from being produced by both the :EGDOC tag and the epifile. This technique is discussed in "Special Techniques" on page 13.

2. The .DD [Define Data File-id] control word to define the DSMUTWTF file is different for the CMS and the TSO environments. SYSVAR "W" files can only be created in these two environments. The file name is in &SYSVARW.

   The macro ends immediately if the operating system is not CMS or TSO because SYSVAR 'W' and SYSVAR 'R' are valid only in these two environments.

3. The DSMUTWTF file is erased to make sure it is empty to start with.

4. The next step is to write some label information out to the file to record what the date is, what file was being formatted[35], and what parameters were specified on the SCRIPT command. The label lines are all comment lines starting with ".*."

   The first line written contains "SCRIPT/VS," the release number (&$DCF), "DEVICE" (&LL@device), and the logical device used (&$LDEV). The subsequent lines of the label contain the parameters which were on the command line. These are contained in the &$PARM symbol. Since &$PARM may be very long it is broken down into 64 character segments for writing out.

5. Four sets of arrays have been used to contain the information to be written out. These are the &@writ@d, &@writ@f, &@writ@h and &@writ@n arrays. Each line of the array contains a call to a macro with an id as the parameter. The macros are DSM#WRTD, DSM#WRTF, DSM#WRTH and DSM#WRTN macro.

   The array separator is redefined to be the characters necessary to construct a symbol name. The &@ symbol is set to a semicolon which is the control word separator.

   Each array is then "issued" to produce a section of the SYSVAR "W" file.

   The array separator is reset to its default value.

---

[35] The file name is extracted from &$PARM using the SUBSTR function of the .SE [Set Symbol] control word. The trailing blanks are stripped from it by setting the symbol &@fnam to itself.

DSM#WRTH

The DSM#WRTH macro writes out entries to the SYSVAR "W" file. Each line written out is a .SE [Set Symbol] control word. The parameter passed to this macro is a heading id.

Two lines are written out for each id—the first line will set the &h1@&*id symbol to the text of the heading and the second line will set &hP@&*id to the page number of the heading.

DSM#WRTF

The DSM#WRTF macro writes out entries to the SYSVAR "W" file. Each line written out is a .SE [Set Symbol] control word. The parameter passed to this macro is a figure id.

Two lines are written out for each id—the first line will set the &f1@&*id symbol to the figure number and the second line will set &fP@&*id to the page number of the figure.

DSM#WRTN

The DSM#WRTN macro writes out entries to the SYSVAR "W" file. Each line written out is a .SE [Set Symbol] control word. The parameter passed to this macro is a footnote id.

The line written out for each id will set the &n1@&*id symbol to the footnote number. In the case of footnote there is no need to save the page number as it is not used in references to the footnote— only the number is.

DSM#WRTD

The DSM#WRTD macro writes out entries to the SYSVAR "W" file. Each line written out is a .SE [Set Symbol] control word. The parameter passed to this macro is a list item id.

Two lines are written out for each id. The first line will set the &d1@&*id symbol to the list item identifier. The second line will set &dP@&*id to the page number that the list item is on.

Modifications to Cross References

Default to Not Print the Cross Reference.

The default for the starter set is to print the cross reference listing and the imbed trace unless specifically suppressed with &SYSVARX on the command. To change this default to not print the cross reference listing, you will have to change the way SYSVAR "X" is processed in the DSM#SETV macro. SYSVAR "X" is processed as follows:

```
.se *a = index '-YES-NO' '-&U'&SYSVARX.'
.if &*a eq 0 .se *a = 1
.se SYSVARX = substr 'yes no' &*a 3
```

The middle line is the one that establishes the default. If SYSVAR "X" wasn't specified on the command or was specified as something other than "yes" or "no," then the default is "yes" because we set &*a to 1 which will cause the last line to pick up the "yes" from the substring subject. To change the default, change the second line shown here to set *a to 5 instead of 1.

Miscellaneous

General Service Macros

There are general service macros within the starter set that do not relate to a single functional area. These macros provide various functions required by several other macros. They may also be used by any user-written APFs or macros.

DSM#CNTX

Certain tags within the starter set are not valid unless particular text structures are going. For example, the :FIGCAP and :FIGDESC tags are not valid unless a figure is open—meaning that a :FIG tag has been encountered but the :EFIG tag has not been encountered yet. In cases such as this we do not want to honor the tag.

One way to handle this would be to simply cancel the tag-to-APF mapping with another .AA [Associate APF] control word. However, this would cause SCRIPT/VS to search for an APF with the same name as the tag. If it cannot find one, an error message is issued indicating that the APF for the tag cannot be found and the tag is treated as text.

The solution we chose to use in the starter set was to map these "invalid" tags to a special macro named DSM#CNTX. This way you will get a more meaningful message and the tag will not appear as text.

The DSM#CNTX macro calls the DSM#MSG macro to issue a message. It passes two parameters to DSM#MSG—the message number "2" and the &$TAG symbol which contains the GML tag which caused the DSM#CNTX macro to be processed.

One of two error messages is issued. The most common message is that the tag found is "out of context," such as an :ALINE tag found outside of an ADDRESS structure. The message macro (DSM#MSG) has logic built in to recognize the tags that the starter set maps to the DSM#CNTX macro. If any other tags are mapped to DSM#CNTX and not recognized by the DSM#MSG macro, a different message is issued that simply shows the tag and says it is unassigned.

DSM#DUPL

The DSM#DUPL macro is called from several macros to advance to the top of an "odd" page without starting the page when duplexing or to the top of the *next* page when not duplexing. It is called by

DSMABSTR
DSMAPPD
DSMBACKM
DSMBODY
DSMFLIST
DSMFRONT
DSMHEAD0
DSMHEAD1
DSMINDEX
DSMPREF
DSMTOC

to get to a new page.

The DSM#DUPL macro performs the following processing:

1. If &$PN is zero, it means that the first page of the document has not been started yet and if so, the macro ends immediately.
2. Any floats that are pending, such as top or bottom figures, are dumped out before starting a new page.
3. The next two lines of the macro undefine themselves the first time they are performed. This is a technique used to do something only once. See "Special Techniques" on page 13 for details on how it works. In this case, the first of the two lines sets a local symbol (&*a) to the macro line number of the second line down from it. It points to the line that reads

   ```
   .if SYSPAGE eq ODD .pa
   ```

   The next line tests to see if duplexing is in effect (&SYSVARD = yes). If it isn't, the macro line pointed to by &*a is deleted from the macro.

   This permanently removes the line from the macro. The next time the macro is executed, the line after

   ```
   .fl dump
   ```

   will be

   ```
   .pa nostart
   ```

   because the two lines that start with the .DM [Define Macro] control word and the line that starts with the .IF [If] control word are all removed from the macro.

   If duplexing is in effect, the .IF [If] control word line remains in the macro.
4. If duplexing is in effect and SYSPAGE is "odd," a page eject is performed to get to an even page. Remember, we're trying to get to the beginning of an odd page. If we're on an even page, all we need to do is the one page eject at the end of the macro, but if we are on an odd page, we must first get to an even page. Then when we do the final page eject, we'll be at the beginning of an odd page.
5. A page eject with the NOSTART parameter is executed to end the current page without starting a new page.

DSM#MSG

All starter set error messages are issued by the DSM#MSG macro. This macro expects various parameters to be passed to it. The first parameter, which is required, is the message number. If no message number is given, message 0 will be issued which states that there was an unassigned error message. It also shows the tag (&$TAG) that prompted the message.

This macro contains the text of the messages. The variable portion(s) are passed in as parameters. The specific parameters passed depend on which message is to be issued. For example, message 3 expects 2 parameters—the type of list being ended and the name of the tag that is ending it. Message 14 expects only one parameter—the index entry level number.

The DSM#MSG macro processes messages as follows:

1. Branches to the appropriate section based on the message number, which is always the first parameter (&*1.)
2. Constructs the message in a local symbol, &*a.
3. Takes another branch to the last section of the macro which will issue the message.
4. Puts the page number into a local symbol and the message number is padded with leading zeroes so that message 1 prints as message "001," which looks better.
5. Issues the message using the .MG [Message] control word. &X'00 is used as a delimiter and "DSMGML" is prefixed to the message number to make it conform to the SCRIPT/VS format. A "W" is appended to the message number to indicate that it is just a warning message. The page number is included after the text of the message.

DSM#RSET

The DSM#RSET macro is called from many other macros to ensure that there is not an open list, footnote, quotation, title page, figure, or example. It is used primarily by document section macros and heading macros to get a "fresh start."

The DSM#RSET macro is called with a single parameter which consists of the variable portion of a constant literal symbol. For example, the DSMPREF macro calls DSM#RSET with a parameter of "Pref." The DSMTOC macro calls DSM#RSET with a parameter of "ToC." There is a symbol named "LL@Pref" which has a value of "Preface." There is also a symbol named "LL@ToC" which has a value of "Table of Contents." The parameter passed is the second part of the symbol name, and the "LL@" portion is constant.

The DSM#RSET macro does not use these parameters or symbol names directly. However, if it becomes necessary to issue an error message during the resetting process, the parameter is passed to the DSM#MSG macro to enable it to construct the appropriate symbol name for the contents of the message.

If none of the elements listed above are currently "open" or "on," the macro simply performs a break (.BR [Break]) and ends. If one of these elements is open, we'll have to find out what it is and end it.

The DSM#RSET macro performs the following processing:

1. DSM#RSET causes a break.
2. The nesting level variables for quotations and lists are &@nest@q and &@nest@l. These variables are incremented each time a list or quotation is begun and are decremented when each is closed. If there is a quotation or list that hasn't been ended yet, the appropriate variable will be greater than one.

   The &@state variable is used to indicate that a figure, footnote, example, or title page is currently open.

These three variables are checked for indications of a structure that needs to be closed or ended before continuing. If &@nest@q, &@nest@l, and &@state are 0, 0, and open, respectively, we are done with the macro.

3. Once we establish that something is open, we have to find out what it is and close it. First the &@nest@q symbol is checked. If it is greater than zero, we issue message 3 and call the DSMEQUOT macro to close the quotation. This will decrement the &@nest@q symbol, but there may have been more than one level of quotation going, so we loop back and check it again. This process will continue until the &@nest@q symbol equals zero, indicating all quotations have been closed.

4. If the list nesting level (&@nest@l) is greater than 0, message 3 is issued and the DSMELIST macro is called. We'll keep looping through this until all levels of lists have been ended (&@nest@l equals zero).

5. If &@state is not equal to "open," it means that a figure, example, title page, or footnote has not been ended. Message 3 is issued and the appropriate ending macro is called depending on the specific value of &@state. These include the DSMEXMP, DSMEFIG, DSMEFTNT, and DSMETTLP macros.

   The value of &@state will have been set specifically by whatever structure is open. For example, the macro that begins figures (DSMFIG) sets &@state to "F." Because all these structures are mutually exclusive, only one can be open at a time. Therefore, we do not need to loop back and re-check &@state.

DSMPSC

The :PSC tag is processed by the DSMPSC APF. :PSC is used for conditional processing. It always uses section number 9 on the .CS [Conditional Section] control word.

1. Turns off the conditional section (9) in case an :EPSC tag was omitted. If we don't do this, we may get a SCRIPT/VS message when we attempt to start section 9 because it may already be started. To avoid this, the section is ended first.

2. Includes conditional section (9). This is the default if there is no PROC attribute on the :PSC tag.

3. Calls DSM@PROC to process the PROC attribute. There are also two pseudonyms for "PROC" which are recognized. These are "PROCESS" and "P." If none of these attribute names are specified, section 9 stays included and is processed. If there is a PROC attribute and it doesn't match the specified process names, the section will be ignored. (See the description of the DSM@PROC macro below.)

4. Starts conditional section 9. The ON parameter on the .CS [Conditional Section] control word is used to begin the definition of what is in the conditional section. The INCLUDE and IGNORE parameters control whether or not what is in the section will be formatted.

DSM@PROC

The DSM@PROC macro processes the PROC attribute of the :PSC tag as follows:

1. Because a list of process names may be passed to this macro, a loop is used to compare each parameter against a list. The list used in the compare includes the logical device name (&$LDEV), the physical device name (&$PDEV), and the value of &SYSVARP which may have been given on the SCRIPT command.

2. When a parameter matches up (&*@ is greater than 0), the macro ends.

3. If no match is found, a .CS [Conditional Section] IGNORE control word is issued to ignore conditional section number 9. The DSMPSC macro, which calls DSM@PROC, set section

4. If the list nesting level (&@nest@l) is greater than 0, message 3 is issued and the DSMELIST macro is called. We'll keep looping through this until all levels of lists have been ended (&@nest@l equals zero).

5. If &@state is not equal to "open," it means that a figure, example, title page, or footnote has not been ended. Message 3 is issued and the appropriate ending macro is called depending on the specific value of &@state. These include the DSMEXMP, DSMEFIG, DSMEFTNT, and DSMETTLP macros.

   The value of &@state will have been set specifically by whatever structure is open. For example, the macro that begins figures (DSMFIG) sets &@state to "F." Because all these structures are mutually exclusive, only one can be open at a time. Therefore, we do not need to loop back and re-check &@state.

DSMPSC

The :PSC tag is processed by the DSMPSC APF. :PSC is used for conditional processing. It always uses section number 9 on the .CS [Conditional Section] control word.

1. Turns off the conditional section (9) in case an :EPSC tag was omitted. If we don't do this, we may get a SCRIPT/VS message when we attempt to start section 9 because it may already be started. To avoid this, the section is ended first.

2. Includes conditional section (9). This is the default if there is no PROC attribute on the :PSC tag.

3. Calls DSM@PROC to process the PROC attribute. There are also two pseudonyms for "PROC" which are recognized. These are "PROCESS" and "P." If none of these attribute names are specified, section 9 stays included and is processed. If there is a PROC attribute and it doesn't match the specified process names, the section will be ignored. (See the description of the DSM@PROC macro below.)

4. Starts conditional section 9. The ON parameter on the .CS [Conditional Section] control word is used to begin the definition of what is in the conditional section. The INCLUDE and IGNORE parameters control whether or not what is in the section will be formatted.

DSM@PROC

The DSM@PROC macro processes the PROC attribute of the :PSC tag as follows:

1. Because a list of process names may be passed to this macro, a loop is used to compare each parameter against a list. The list used in the compare includes the logical device name (&$LDEV), the physical device name (&$PDEV), and the value of &SYSVARP which may have been given on the SCRIPT command.

2. When a parameter matches up (&*@ is greater than 0), the macro ends.

3. If no match is found, a .CS [Conditional Section] IGNORE control word is issued to ignore conditional section number 9. The DSMPSC macro, which calls DSM@PROC, set section 9 up to be included. We need to override this if a PROC attribute value is present but not valid.

DSMIM

The DSMIM macro is used to process the .IM [Imbed] control word to imbed a file. It produces a message stating that the file is being imbedded and saves the file name for the imbed trace that is printed with the cross reference listing.

The .IM [Imbed] control word is "mapped" to DSMIM by the DSM#SET macro during initialization. This is done by defining a macro named "IM" that calls the DSMIM macro and passes all the control word parameters along as macro parameters.

```
    .dm im /.dsmim &*
```

Since .IM [Imbed] is a control word and not a GML tag, it can not be overridden except by a macro which has the same name as the control word. All of the permanent macros included in the starter set macro library must start with "DSM," which means that we have had to dynamically define a macro with the same name as the control word (IM). This IM macro simply passes the control word parameters along to the DSMIM macro for processing. This additional level of indirection is necessary to maintain the naming scheme for the macro library.

The file name is the first parameter passed to this macro. The DSMIM macro performs the imbed in the following way:

1. The file name is compared to the names of the utility files. If a match is found, the file is imbedded and the macro ends. Utility files are not included in the imbed trace or in the messages, so there is nothing else to be done.

2. The &@nest@i symbol indicates the level of nesting of imbedded files. The symbol is incremented each time a file is imbedded. It will be decremented later as described below.

3. Then we define a local symbol (&*a) to be three times the level of imbed nesting. This will be used to redefine &*a to be a string of dashes. Because &*a is used as the length parameter in the substring function, the number of dashes will be the nesting level times 3. Therefore, the names of files imbedded from other files will appear indented.

```
(Pass 1)  Page   10: Imbedding file ---> First
(Pass 1)  Page   10: Imbedding file ------> Second
(Pass 1)  Page   10: Imbedding file ---------> Third
```

4. The page number is put into a local symbol and padded with blanks so that it will always be five spaces wide. This will be used in the imbed message.

5. If the system symbol &$TWO is 1, it indicates that we will be doing more than one pass. This means we need to include the pass number in the message. The &*pass symbol is defined to contain the following information enclosed in parentheses:

   - The character string "Pass" which is in the &LL@Pass symbol
   - The pass number, which is in the &$PASS symbol.

   When only one formatting pass is been processed, the pass number is not necessary. We don't need to define the &*pass symbol because local symbols that haven't been defined have a null value in a macro.

6. A message is issued that includes the following pieces of information.

   - The word "Pass" and the pass number (&LL@Pass &$PASS)
   - The word "Page" (&LL@Page) and the page number (&*page)
   - A colon (:)
   - The word "imbedding" (&LL@Imbdg)
   - Some dashes (*a)
   - An arrow end (>)
   - The full file name (&*) in uppercase letters.

7. If this isn't the last pass (indicated by &@lastpass not equal to "yes"), then we don't need to bother with saving the information for the imbed trace. This will be done on the last pass. In this case, we skip over this code and go directly to imbedding the file. (See number 9. below.)

8. If this is the last pass:

   a. A local symbol (&*a) is defined to be two times the level of imbed nesting. Then it is set up to be a string of blanks that is twice as long as the the nesting level.

   b. An entry is made into the &@imtrace array. This array contains the imbed trace information that will be printed along with the cross reference at the end of the document. The entry includes:

      - The word "Page" (&LL@Page)
      - The page number (&*p)
      - A tab character (&$TAB)
      - Some blanks (&*a)
      - The file name (&*) in uppercase letters.

9. The file requested is then imbedded. The imbed nesting counter &@nest@i is then decremented to show that the imbed was completed.

DSMEPSC

The :EPSC tag is processed by the DSMEPSC APF which ends the conditional section that was started by the :PSC tag. The section number is 9.

Modifications to General Service Macros.

Because the macros described in this chapter are used throughout the starter set to provide general services to APFs, any modifications to them should be approached with caution. Some simple modifications to incorporate local tags and structures can safely be done and are described below.

Adding New Messages

In adding your own tags and APFs to the starter set, you may find it necessary to generate warning messages for your users. These should be put in DSM#MSG to take advantage of the fact that all messages are collected there. To add a new message, pick a number not already used and add whatever lines are needed to generate the text of the message in the &*a local symbol. For example, you could add the following DSM#MSG call to your APF:

```
.dsm#msg 15 Part number
```

You would also need to add the following lines to DSM#MSG to actually issue the message:

```
...msg15
.se *a '&*2 &*3 is invalid - truncated to 8 characters'
.go mg
```

Eliminating the Imbed Trace

Although most users find the imbed trace useful, you might want to disable it. If so, remove the line in the DSM#SET macro that defines the IM macro that gets control instead of the .IM [Imbed] control word.

```
    .dm im /.dsmim &*
```

This disables both the trace at the end of the document and the messages that appear on the screen as the files are imbedded.

To disable only the trace, add a .ME [Macro Exit] control word to the DSM#XLST macro after the "...imtr" label.

```
   ...imtr
   .me
```

See the discussion of producing the imbed trace in "Cross-References" on page 147 for more details on this process.

Appendix A. Modifying the Macros

SCRIPT/VS and the starter set operate in several different operating system environments. If you want to modify a macro or create a new macro, what you need to do depends on what environment you are operating in. The four sections that follow provide quick instructions on how to modify the maclib in the four environments–CMS, TSO, DLF and ATMS.[36]

These instructions are not meant to be all inclusive. It is assumed that you have already mastered the basic skills of working on the system you use. All that is provided here is some special information you may need to work with the macros. You should consult the system documentation for additional information and assistance.

Modifying the DSMGML3 MACLIB (CMS)

In the CMS environment the starter set macros are kept in a file named DSMGML3 MACLIB. In order to modify it you must have write-access to the disk that it is stored on or you must copy it to your own disk. *The maclib cannot be modified directly by most editors.*

In order to modify a macro you will need to:

1. Extract the member from the maclib
2. Edit it
3. Replace it in the maclib.

Note: Several of the macros in the starter set contain the pound sign (#) character or the at-sign (@) character. These are the default line-end and character-delete characters in CMS. When you attempt to work on these macros you will need to change these terminal settings to use other characters. For example, the following command changes the line-end character to a semicolon (;) and the character-delete character to a (%):

```
term linend ; chardel %
```

Extracting a Member from a Maclib

There are several different ways to extract a member from a maclib. One way is:

1. Define two files one of which is the member of the maclib that you want extracted. The other is the name of the file where you want it put.

   ```
   filedef inmove disk dsmgml3 maclib a (member dsmlistm
   filedef outmove disk dsmlistm copy a
   ```

[36] The GML starter set is a fully supported part of the Document Composition Facility program product provided that neither the profile nor the macro library have been modified in any way. This means that you should be careful to not alter the base version of these files. Make your own copies or user libraries instead.

The last name on the first line is the name of the macro you are interested in. The file named on the second line should have the same filename as the macro and should have a filetype of 'COPY'. "inmove" and "outmove" are the default file identifiers for the MOVEFILE command[37].

2. Next, request that the 'inmove' file be moved to the 'outmove' file.

   ```
   movefile
   ```

   This will create a file named 'DSMLISTM COPY' on your A-disk.

   Make sure that before executing this step that you don't already have a file with the same name as the 'outmove' file[37]

3. Then, using the editor of your choice, edit the new file

   ```
   xedit dsmlistm copy
   ```

4. Copy files usually have special characteristics in CMS which you will have to work around. The editor default is usually to put sequence numbers on the right-hand side of the lines. We can't let it do that, so before we do anything we need to inhibit the numbers. To prevent sequence numbers in XEDIT, use the following command:

   ```
   serial off
   ```

   In other editors, the command may be different.

   Also the default case for copy files is usually uppercase. The GML starter set macros are written in mixed case. To avoid getting the file changed to uppercase, use the following command:

   ```
   case mixed
   ```

Editing the Macro

Now edit the file to make the changes you want to make and file it away.

Creating Your Own Maclib

The first time you create a new macro or modify one of the ones in the starter set, you will need to create a new macro library of your own. Be careful not to modify the starter set library in any way.

You can generate your own library with the following command which generates a new maclib named "USER" and puts the macro named "DSMLISTM" into it:

```
maclib gen user dsmlistm
```

The name of the CMS command is "MACLIB" and "GEN" is a parameter on the command.

[37] See the CMS Command documentation for more information about the FILEDEF and MOVEFILE command.

Using Your Own Maclib

The LIB option of the SCRIPT command controls which libraries SCRIPT/VS will search for macros and in what order. By specifying both the name of your own maclib and the starter set maclib (DSMGML3) with the LIB option you can get SCRIPT/VS to use your macros ahead of the starter set macros.

```
scriptvs mydoc (prof(dsmprof3) lib(user dsmgml3)
```

See *Document Composition Facility: SCRIPT/VS Language Reference* for additional explanation of the LIB option.

Replacing the Macro in the Maclib

For subsequent modifications to macros or new macros you will need to use the MACLIB command again with a different parameter. The "ADD" parameter will add a new macro to your library:

```
maclib add user dsmbody
```

The "REP" parameter will replace a macro that already exists in the library with a new copy of it.

```
maclib rep user dsmbody
```

Creating a New Macro

To create a new macro, all you need to do is create a file whose filetype is "COPY." Again, make sure that it doesn't have sequence numbers in it. To add a new macro to the macro library use the following command:

```
maclib add user mymac
```

where "mymac" is the filename of the new macro and "user" is the name of the macro library.

Compressing the Maclib

Each time you replace a member in the maclib there will be some unused space left in the file where the macro used to be. This means that the size of maclib will keep increasing each time you replace a macro. There is a simple way to compress the maclib and eliminate all the extra space in it.

```
maclib comp user
```

That's all you have to do.

Modifying SCRIPT.R30.MACLIB (TSO)

The GML starter set is stored in a partitioned data set (PDS) in TSO. Each macro is a separate member of the PDS and as such can be created and edited in the same way that all PDS members are edited using the editor of your choice. The starter set library is named

"SCRIPT.R30.MACLIB" and can be used as a source for macros which you wish to modify and then save in your own library.[38]

New and modified macros should be placed in a separate macro library, not replaced or added to the starter set macro library. The name of the new library should be different from the name for the starter set library.

If you choose to modify the starter set library directly, make sure that you have saved a complete starter set library elsewhere first. Any problems you report must be reproducible on an unmodified starter set library. It is generally a better idea to create a separate user library and format using that library and the starter set library. How to do this is explained below.

Using Your Own Maclib

The LIB option of the SCRIPT command can be used to provide the name of a library for SCRIPT/VS to search for macros and symbols before searching the SCRIPT.R30.MACLIB.

```
     scriptvs mydoc (prof(dsmprof3) lib('user.script.maclib')
```

In the command shown above, SCRIPT/VS will automatically concatenate SCRIPT.R30.MACLIB to the library that was specified with LIB.

See *Document Composition Facility: SCRIPT/VS Language Reference* for additional explanation of the LIB option.

Creating a New Macro

To create a new macro, all you need to do is create a new member in your user maclib.

Modifying DSMGML3 Macros (DLF)

The GML starter set macros and profile are stored as documents in public library number 1314151 in the DLF environment.[39] To modify one of these documents:

1. EXPORT it from 1314151 into a partitioned data set in TSO
2. Edit it using your favorite editor (such as ISPF)
3. File it
4. IMPORT it back into DLF.

You can either import the file back into 1314151, if you are authorized to do so, or you can import it into your own user library or your project library. In order to maintain support of the macro library, it is better to create your own project or user library than to modify the 1314151 library.

To create a new macro, all you need to do is import a new document into your user library.

You don't need to do anything else to format using your new macros. SCRIPT/VS will automatically search your private library, your project library and the 1314151 library (in that order) when it is searching for macros and symbols that are in a library. See also *Document Library Facility Guide* for information about using the SCRIPT command in DLF.

[38] This is the default library name which may have been changed by your installation.

[39] This is the default library which may have been changed by your installation.

Modifying DSMGML3 Macros (ATMS)

In ATMS-III the GML macros and profile are stored in the permanent storage of the SYSOP operator. The prefix for the documents is "DSM30" for the Document Composition Facility Release 3 APFs and profile. The default SYSOP operator number is 5.

To modify one of the GML macros:

1. Obtain a copy of the macro from SYSOP's permanent storage:

   ```
   gc;DSM30DSMLISTM:5
   ```

   Note: You can also create your own new macros and store them in your own permanent storage.

2. Edit the macro and, using a prefix other than DSM30, store it in your own permanent storage.

3. Build and connect an index for the prefix you use.

4. If you are going to use the GML starter set and your own macros at the same time, you will also need to build and connect that library.

When the two libraries are connected, SCRIPT/VS will be able to search both for the macros it needs. The order in which you build and connect the libraries is important. SCRIPT/VS will search the indexes in the order you built them. So, if you are using your library (containing copies of some of the starter set macros), build and connect your library first, then build and connect the DSM30 library.

If the indexes are built in the wrong order, use the LIB option of the SCRIPT command to specify the search order.

```
LIB(opnum [...opnum")
```

See the ATMS-III documentation for more details on how to do all this.

Appendix B. Migration from Release 2 to Release 3

The purpose of this section is to describe the modifications made to the Release 2 starter set for Release 3 of the Document Composition Facility. Here we provide some guidelines for updating APFs and macros that you wrote for Release 2. If you do not plan to use a 4250 printer, IBM 3820 Page Printer, or a 3800 Printing Subsystem Model 3 in all-points addressability mode, you probably won't have to change your GML.

All of the macros in the starter set macro library have been modified for Release 3 of Document Composition Facility. In some cases the change is nothing more than changing the name of the macro to begin with the letters "DSM." In other cases substantial revisions were made to the logic and function of the macros. Some macros have been deleted for Release 3 with their functions absorbed by other macros or the profile. If you modified some of the Release 2 macros, you need to carefully review each modification and compare the Release 2 macro to the Release 3 macro. Changes you made might work without modification in Release 3 or they might need to be modified.

We had a number of reasons for changing the macros:

- To formally support the starter set, the naming convention for the macros was standardized to include the three character prefix indicating DCF
- To make it easier for you to tailor the starter set, many functions (particularly definitional things) were moved into the profile
- To support page printers, some processing functions were modified.

This last category of changes requires additional explanation and should be used as a guide to reviewing your own GML. The functional areas where we made changes include:

- Providing font support for page printers (4250 printer, 3800 Printing Subsystem Model 3, and IBM 3820 Page Printer)
- Working in device units rather than unqualified space units
- Using system symbols.

Font Support for Page Printers

Since a greater variety of fonts can be used with page printers we changed the starter set to take advantage of the font capabilities of each machine. This meant reviewing each text element in the starter set and selecting a suitable font for it. For example, each piece of information placed on the title page was considered separately; many were given special font definitions. These definitions were placed in DSMPROF3 to make it easy to override them. For example,

```
.df title type (24 bold) up
.df author type(12
.df address type(10)
.df date type(11 italic)
.df docnum type(10 italic)
.df titlesec type(10 italic bold)
```

When the title page is formatted, the .BF [Begin Font] control word is used to change fonts for the page printers without disturbing the font that is used for line devices. For example:

```
    .bf docnum =
```

starts a font named "docnum" that is defined for the page printers only. For all other devices, the current font will be started again.

Using Device Units

In several instances in the starter set it was also necessary to adjust the space unit specifications. Whenever space units were given in unqualified horizontal or vertical space unit notations such as,

```
    .sp 4
```

or

```
    .in 6
```

each one needed to be carefully reviewed. For line devices, unqualified horizontal space units are interpreted as ordinary word spaces which are the same width as all of the characters. However, space unit notations of this type cause problems on page printers because unqualified space units are interpreted as figure spaces in the default font (roughly the width of the character zero). These spaces cannot be equated to a number of characters since each character in a font for page printers has its own width. A figure space is far smaller than most of the characters in the font. Therefore, if you set a delayed indent of 6 characters to indent the second line to after the sixth character on the first line (just fine for a line device), it would not work out right on a page printer.

In the starter set, some of the indention values were adjusted to provide the correct result for page printers. In some cases the width of the indention had to be carefully measured. This was the case in the DSMFCAP macro which in Release 2 used to set an indent of 10. The 10 was equivalent to the width of "Figure 1. ." In Release 3 we had to measure this width exactly in device units to set the indention. You can see the lines that perform the measurement and set the indention in the DSMFCAP macro. These lines are explained in "Examples and Figures" on page 115.

Another example of where unqualified space notation had to be changed is in the DSM#SETX macro. It sets up the tab rack for the cross reference listing. In Release 2 tabs were set like this:

```
   .tb 9 19 27 32
```

In Release 3, these tab positions were too small for page printers. Since precision here was not important as long as the lines all fit on the page, the tabs were changes to

```
   .tb 9m 19m 27m 32m
```

to provide more space when formatting for page printers.

Using System Symbols

Another type of problem arises from using certain system symbols which can be set very precisely, but return rounded values.

The easiest way to explain the rounding problem is to look at what happens in Release 2 and compare it to the Release 3 result. The Release 2 LIST macro (equivalent to the DSMLISTM macro in Release 3) sets this symbol value:

```
.se @in@l = &$IN
```

If the indention is "4", the &@in@l symbol is set to 4. However, suppose the indention was ".3i." In Release 2, the resulting value of &@in@l depends on what device you are formatting for:

- For a terminal, a 1403, or a 3800 with 10-pitch fonts the answer is 3.
- For a 3800 with a 12-pitch font the answer is 4.

In either case, no rounding is done.

In Release 3, the problem is expressed in terms of what the result is for page printers. One-third of an inch should result in 200 device units for a 4250 printer and 80 for a 3800 Printing Subsystem Model 3 and IBM 3820 Page Printer. However, the value of system symbols is given in figure spaces for page printers. And, since a figure space in a 10-point normal font is 42 device units, the value of &$IN will be rounded to "7" which is off by 6 device units from the original specification.

The way to avoid this rounding is to request the value of the system symbol in device units using the &DH′ or &DV′ symbol attributes:

```
.se @in@l = &dh'&$IN.dh
```

The value for line printers will be the same as before and the value for the 4250 printer will have been properly set at 200 device units. For the 3800 Printing Subsystem Model 3 and IBM 3820 Page Printer, the value will be 80.

Appendix C. Starter Set Macro Library Listing

MACRO NAME	PAGE	MACRO NAME	PAGE	MACRO NAME	PAGE
DSM#CNTX	182	DSMADDR	198	DSMHEAD2	211
DSM#DUPL	182	DSMALINE	198	DSMHEAD3	211
DSM#LINT	182	DSMAPPD	198	DSMHEAD4	211
DSM#LTYP	183	DSMAUTHR	198	DSMHEAD5	212
DSM#MSG	184	DSMBACKM	199	DSMHEAD6	212
DSM#RSET	185	DSMBODY	199	DSMHP0	212
DSM#SET	185	DSMCIT	199	DSMHP1	212
DSM#SETS	186	DSMDATE	199	DSMHP2	212
DSM#SETV	187	DSMDCNUM	199	DSMHP3	212
DSM#SETX	187	DSMDDEF	200	DSMIDMMY	212
DSM#STYL	188	DSMDDHD	200	DSMIEH	213
DSM#SUPR	189	DSMDLIST	200	DSMIHD1	213
DSM#TIPG	190	DSMDTERM	201	DSMIHD2	213
DSM#WRIT	191	DSMDTHD	201	DSMIHD3	214
DSM#WRTD	191	DSMEADDR	201	DSMIM	214
DSM#WRTF	191	DSMECIT	201	DSMINDEX	214
DSM#WRTH	191	DSMEFIG	202	DSMINDX1	215
DSM#WRTN	191	DSMEFTNT	202	DSMINDX2	215
DSM#XLST	192	DSMEGDOC	202	DSMINDX3	215
DSM#XRFD	193	DSMEHP	202	DSMIREF	216
DSM#XRFF	193	DSMELIST	203	DSMLIREF	217
DSM#XRFH	193	DSMELQU	204	DSMLISTM	218
DSM#XRFI	193	DSMEPSC	204	DSMLITEM	219
DSM#XRFN	194	DSMEQUOT	204	DSMLPART	219
DSM#YESN	194	DSMETTLP	204	DSMLQUOT	219
DSM@FRME	194	DSMEXMP	205	DSMNOTE	219
DSM@IDS	195	DSMFCAP	205	DSMOLIST	219
DSM@IPRT	195	DSMFDESC	206	DSMPARA	220
DSM@MAC@	195	DSMFGREF	206	DSMPARA1	220
DSM@PGRF	195	DSMFIG	207	DSMPARA2	220
DSM@PLCE	196	DSMFLIST	207	DSMPARA5	220
DSM@PROC	196	DSMFNREF	208	DSMPCONT	220
DSM@RFID	196	DSMFRONT	208	DSMPREF	220
DSM@RIDI	196	DSMFTNT	209	DSMPROF3	221
DSM@SEC	196	DSMGDEF	209	DSMPSC	227
DSM@SEE	196	DSMGDOC	209	DSMQUOTE	227
DSM@SEEI	197	DSMGLIST	209	DSMSLIST	228
DSM@SHD	197	DSMGTERM	209	DSMTITLE	228
DSM@STTL	197	DSMHDREF	210	DSMTOC	228
DSM@WIDT	197	DSMHEAD0	210	DSMTTLEP	228
DSMABSTR	197	DSMHEAD1	211	DSMULIST	228
				DSMXMP	229

The following IBM copyrighted program code may be reproduced and distributed by the customer to other IBM customers who are licensed to use the Document Composition Facility program.

DSM#CNTX

```
.* DSM#CNTX: Tag out of context are mapped here to give a message.    *
.*********************************************************************
.dsm#msg 2 &$TAG
```

DSM#DUPL

```
.*  DSM#DUPL:  Advance to before an odd page (duplex) or next page.   *
.*  If no page started, exit, otherwise dump pending floats.          *
.*********************************************************************
.if &$PN eq 0 .me
.fl dump
.* DUPLEX - EJECT TO EVEN PAGE. NOT DUPLEX - REMOVE THE EJECT LINE    *
.dm dsm#dupl(&$LNUM.) off &$CW..se *a = &$LNUM + 20
.dm dsm#dupl(&$LNUM.) off &$CW..if &SYSVARD eq no .dm dsm#dupl(&*a.) off
.if SYSPAGE eq ODD .pa
.pa nostart
```

DSM#LINT

```
.* DSM#LINT:  No parms. Internal service macro to define nesting      *
.* control symbols for ordered and unordered list.                    *
.* &@denest@u and &@renest@u are ring counters for unordered lists.   *
.* &@denest@o and &@renest@o are ring counters for ordered list       *
.*********************************************************************
.se @denest@u = substr &L'&@ulistnest.12345678 1 &L'&@ulistnest
.se @renest@u = substr &@denest@u.&L'&@ulistnest.1 3 &L'&@ulistnest
.se @denest@o = substr &L'&@olistnest.12345678 1 &L'&@olistnest
.se @renest@o = substr &@denest@o.&L'&@olistnest.1 3 &L'&@olistnest
```

DSM#LTYP

```
.* DSM#LTYP: Parm = ALPHA, BULLET, NUMBERED,ORDERED,ROMAN, SIMPLE,      *
.* UNORDERED.  Used by DSMLISTM macro to indicate the type of list to *
.* be formatted. &@ltype is set to "s", "o", "u", or "l" to indicate  *
.* the type of list.  &@id@l is set to a string which will generate   *
.* the 'ids' for subsequent list items. This macro is self-modifying. *
.**********************************************************************
.dsm#lint
.*            SET &@ltype TO THE TYPE OF LIST THAT SHOULD BE CREATED    *
.se *a = substr &*l 1 1
.se *b     = index  ' *OSUG' &U'&*a
.if &*b eq 0 .se *b = 1
.se @ltype = substr 'zdosug' &*b 1
.go list&@ltype
.*            FOR DEFINITION LISTS AND SIMPLE LISTS, THE "id" ID NULL   *
...listg
...listd
...lists
.se @id@l = ''
.me
.* &@nest@u INDICATES THE NESTING LEVEL, AND @renest@u CONTAINS A      *
.* RING COUNTER USED TO DETERMINE THE NESTING LEVEL OF THE NEXT UL     *
...listu
.if &E'&@nest@u eq 0 .se @nest@u = &L'&@ulistnest
.se @nest@u = substr &@renest@u &@nest@u 1
.se *a = substr &@ulistnest &@nest@u 1
.go listz
.* &@nest@o INDICATES THE NESTING LEVEL, AND @renest@o CONTAINS A      *
.* RING COUNTER USED TO DETERMINE THE NESTING LEVEL OF THE NEXT OL     *
...listo
.if &E'&@nest@o eq 0 .se @nest@o = &L'&@olistnest
.se @nest@o = substr &@renest@o &@nest@o 1
.se *a = substr &@olistnest &@nest@o 1
.* &*a IS A ONE-LETTER KEY INDICATING THE TYPE OF LIST ITEMID TO USE  *
.* &@id@l IS SET TO A STRING THAT WILL GENERATE THE PROPER IDENTIFIER *
...listz
.se *a = '&@ltype.&*a.
.if &E'&@id@l@&*a eq 0 .dsm#msg 7 &*a
.se @id@l '&V'&@id@l@&*a..'
```

DSM#MSG

```
.* DSM#MSG:  Parm = message #, variables   Issues error messages with *
.* the .MG control word.  The 1st parm is the GML msg #, subsequent   *
.* parms are variables to be substituted into the messages text.      *
.*********************************************************************
.go msg&*1
...msg
...msg0
...msg1
.se *a 'Unassigned error message ... tag is &$TAG
.go mg
...msg2
.if &*2 eq AUTHOR
.or &*2 eq DATE
.or &*2 eq DOCNUM
.or &*2 eq TITLE   .se *a 'Title Page
.if &*2 eq ALINE   .se *a 'Address
.if &*2 eq DT
.or &*2 eq DD      .se *a 'Definition List
.if &*2 eq GT
.or &*2 eq GD      .se *a 'Glossary List
.if &*2 eq FIGCAP
.or &*2 eq FIGDESC .se *a 'Figure
.if &*2 eq LI
.or &*2 eq LP      .se *a 'List
.if &E'&*a eq 0 .go msg1
.se *a '&*2 tag found outside &*a
.go mg
...msg3
.if &*2 eq listd .se *a 'Definition List
.if &*2 eq listg .se *a 'Glossary List
.if &*2 eq listo .se *a 'Ordered List
.if &*2 eq lists .se *a 'Simple List
.if &*2 eq listu .se *a 'Unordered List
.if &*2 eq listz .se *a 'List
.if &E'&*a eq 0 .se *a '&LL@&*2
.se *a '&*a prematurely ended by &LL@&*3 tag
.go mg
...msg4
.se *a '&*2 tag found within &LL@&*3 and ignored
.go mg
...msg5
.se *a 'Extraneous &LL@&*2 Term '&*3.' ignored
.go mg
...msg6
.se *a '&LL@&*2 Term tag missing
.go mg
...msg7
.se *a 'Unrecognized List type: &*2
.go mg
...msg8
.se *a '&LL@&*2 id '&*3.' truncated to seven characters
.go mg
...msg9
.se *a 'Duplicate &LL@&*2 id '&*3.' ignored
.go mg
...msg11
.se *a '&LL@&*2 end-tag found outside &LL@&*2 and ignored
.go mg
...msg14
.se *a 'Missing term for level &*2 index entry
.go mg
.*         PAD THE MSG # WITH LEADING ZEROES & ISSUE MSG WITH PAGE #  *
...mg
.se *p = &
.se *b = substr '00&*1.' &L'&*1 3
.'mg &X'00.DSMGML&*b.W&X'00.&*a.. (Page &*p.)
```

DSM#RSET

```
.* DSM#RSET: Parm = message text. Used by heading and section tags to *
.* end open lists, quotes, figures or footnotes. Causes a break.      *
.*--------------------------------------------------------------------*
.br
.if &@nest@q.,&@nest@l.,&@state eq 0,0,open .me
.*              END ALL QUOTES BY ISSUING MSG & CALLING DSMEQUOT      *
...loop1
.if &@nest@q eq 0 .go loop2
.dsm#msg 3 QtePh '&*.'
.dsmequot
.go loop1
.*              END ALL LISTS BY ISSUING MSG & CALLING DSMELIST       *
...loop2
.if &@nest@l(0) eq 0 .go state
.dsm#msg 3 list&@ltype '&*.'
.dsmelist
.go loop2
.*     CHECK @state FOR "open" FIG, XMP, OR FN. ISSUE MSG & CLOSE IT  *
...state
.if &@state eq open .me
.dsm#msg 3 &@state '&*.'
.if &@state eq Exmpl .dsmexmp
.if &@state eq F   .dsmefig
.if &@state eq N .dsmeftnt
.if &@state eq TtlPg .dsmettlp
```

DSM#SET

```
.*   DSM#SET: Initialization of important symbols.                    *
.**********************************************************************
.se @lastpass = 1 - &E'&@lastpass * &$TWO * 3 + 1
.se @lastpass = substr 'yesno' &@lastpass 3
.*        DEFINE SOME COMMON SYMBOLS TO GET SPECIAL CHARACTERS.       *
.se rbl '&$RB
.* INITIALIZE VARIOUS COUNTERS & STRINGS: &@sk@l = skip between litems*
.* &@nest@l - List nesting          &@nest@i - Imbed nesting          *
.* &@nest@q - Quote nesting         &@fig#   - Figure number          *
.* &@fn#    - Footnote number       &@state  - Fig/Fn/Xmp nesting     *
.gs args 0       0       0       1     1   open   &@sk@ls
.gs vars @nest@l @nest@i @nest@q @fig# @fn# @state @sk@l
.se @nest@o off
.se @nest@u off
.*   INIT CROSS REFERENCE ARRAYS, TAKE OVER .IM, INIT SYSVARW ARRAYS  *
.if &SYSVARX ne yes .go skip2
.dm im /.dsmim &*/
.se @xref@f( ) = '.*'
.se @xref@h( ) = '.*'
.se @xref@i( ) = '.*'
.se @xref@n( ) = '.*'
.se @xref@d( ) = '.*'
...skip2
.if &E'&SYSVARW eq 0 .go skip3
.se @writ@f( ) = '.*'
.se @writ@h( ) = '.*'
.se @writ@i( ) = '.*'
.se @writ@n( ) = '.*'
.se @writ@d( ) = '.*'
.*IF INDEXING, DEFINE &@it1, &@it2, AND &@it3, TO CONTAIN THE 1ST,2nd *
.* & 3RD INDEX TERMS, SO .PI WILL GENERATE A MSG IF ONE IS OMITTED    *
...skip3
.if &$INDX eq 1 .th .gs args ''   ''   ''
.th                 .gs vars @it1 @it2 @it3
.*     DEFINE SYMBOLS TO USE IN RH/RF DEFINITIONS                     *
.se @stitle off
.gs args ''   ''      '&@stitle.'
.gs vars @sec @stitle @shead
.dm dsm#set off
```

DSM#SETS

```
.*  DSM#SETS: Define symbols for general use in starter set.         *
.*********************************************************************
.se LL@Abstr 'Abstract
.se LL@Appdx 'Appendix
.se LL@BkMtr 'Back Matter
.se LL@Body  'Body
.se LL@CrsRf 'Cross Reference
.se LL@Def   'Definition
.se LL@DocNm 'Document Number
.se LL@Exmpl 'Example
.se LL@F     'Figure
.se LL@File  'File
.se LL@N     'Footnote
.se LL@Gloss 'Glossary
.se LL@H     'Heading
.se LL@Imbdg 'Imbedding
.se LL@ImTrc 'Imbed Trace
.se LL@Index 'Index
.se LL@I     'Index
.se LL@List  'List
.se LL@LI    'LI
.se LL@L     'List Item
.se LL@D     'List Item
.se LL@LstIl 'List of Illustrations
.se LL@Note  'Note
.se LL@of    'of
.se LL@onpge 'on page
.se LL@Page  'Page
.se LL@Part  'Part
.se LL@Pass  'Pass
.se LL@Pref  'Preface
.se LL@QtePh 'Quoted Phrase
.se LL@Refs  'References
.se LL@SpChr 'Special Characters
.se LL@Syntx 'Syntax
.se LL@ToC   'Table of Contents
.se LL@Table 'Table
.se LL@TtlPg 'Title Page
.se LL@unkn  'unknown
.se LL@device 'DEVICE
.se LL@format 'Formatted with
.se LL@at    'at
.se LL@saved 'saved on
.*       SET &date TO BE OF THE FORM "July 4th, 1776".              *
.gs args January February March April May June July August
.gs args &* September October November December
.se *s 'stndrdththththththththththththththththstndrdththththththththst
.se *a = &SYSDAYOFM * 2 - 1
.se *a = substr '&*s.' &*a 2
.se *b = &SYSDAYOFM + 0
.se *c = &SYSMONTH + 0
.se date '&*&*c &*b.&*a., 19&SYSYEAR
.*          SET &time TO BE OF THE FORM "11:01 a.m.".               *
.if &SYSHOUR lt 12 .se *m = 'a.m.
.el .se *m = 'p.m.
.if &SYSHOUR le 12 .se *h = &SYSHOUR + 0
.el .se *h = &SYSHOUR - 12
.se time '&*h.:&SYSMINUTE &*m
.dm dsm#sets off
```

DSM#SETV

```
.*  DSM#SETV: Process SYSVARs and set defaults.                        *
.*  See Starter Set Implementation Guide on 'Initialization'          *
.**********************************************************************
.se *a =      index '-NO-YES-DUPLEX-SIMPLEX-' '-&U'&SYSVARD.'
.if &*a eq 0 .se *a = 1
.se SYSVARD = substr 'no yes yes    no' &*a 3
.if &E'&SYSVARH eq 0 .se SYSVARH = no
.se *a =                 index '-NO--YES-NUMBER-' '-&U'&SYSVARH.'
.if &*a ne 0 .se SYSVARH = substr 'no  1.0 1.0' &*a 3
.if &E'&SYSVARP eq 0 .se SYSVARP = ''
.*                                                                    *
.if &E'&SYSVARR ne 0 .an &$SYS eq CMS
.th .dd DSMUTREF &SYSVARR dsmrefs *
.if &E'&SYSVARR ne 0 .an &$SYS eq TSO
.th .dd DSMUTREF dsn &SYSVARR..DSMREFS
.if &$SYS eq CMS .or &$SYS eq TSO .an &E'&SYSVARR ne 0 .im DSMUTREF
.*                                                                     *
.if &E'&SYSVARS eq 0 .se SYSVARS = &SYSVARC
.se *a      = index '-1---2---OFFSET-ONE-TWO-' '-&U'&SYSVARS.'
.if &*a eq 0 .se *a = 1
.se SYSVARS = substr 'one two off    one two' &*a 3
.se *a =      index '-YES---NO----RIGHT-CENTER-LEFT-' '-&U'&SYSVART.'
.if &*a eq 0 .se *a = 1
.se SYSVART = substr 'right no    right center left' &*a 6
.se *a =       index '-YES-NO-' '-&U'&SYSVARX.'
.if &*a eq 0 .se *a = 1
.se SYSVARX = substr 'yes no' &*a 3
.dm dsm#setv off
```

DSM#SETX

```
.*   DSM#SETX:  Parms= Figure, Heading, List item or Footnote         *
.*   Creates cross reference listing header for the type indicated    *
.**********************************************************************
.sk 4
.in
.ir
.kp 2i
.se @xref@of = 32m
.tb 9m 19m 27m 32m
.dc asep , 40
.*          FORMAT A HEADING FOR THE CROSS-REFERENCE LISTING          *
.bf hi2
.bx 1 r
.ce &LL@&*1 ID's
.bx off
.sp 2
.us id&$TAB.&LL@File.&$TAB.&LL@Page.&$TAB.&LL@&*1 &LL@Refs
.pf
.sp 1
```

DSM#STYL

```
.* DSM#STYL:  Parameters = ONE, TWO, OFF, or (null)                 *
.* Establishes page layout style. If no parameter, uses SYSVARS.     *
.*******************************************************************
.if &L'&*1 eq 0 .go &SYSVARS
.el .go &*1
.* 1 PAGE WIDTH COL. THE RC SPACE IS 2 IN BINDING. FNs ARE FULL PAGE  *
...one
.gs args ''   ''   0    0
.gs vars @rc1 @rc2 @fn1 @fn2
.cd 1 0
.cl
.rc adjust
.if &SYSVARD eq yes .dh 0 outside
.th .dh 1 outside
.go fnldr
.*        2 EQUAL SIZED COLS. GUTTER = 4, FNs ARE FULL PAGE          *
...two
.se *cl = &DH'&$LL - &DH'&@gutter / 2
.se *cd = &*cl + &DH'&@gutter
.gs args ''   ''   0    0
.gs vars @rc1 @rc2 @fn1 @fn2
.cd 2 0 &*cd.dh
.cl &*cl.dh
.rc adjust
.if &SYSVARD eq yes .dh 0 outside
.th .dh 1 outside
.go fnldr
.* OFFSET: 1 COL FILLING 4/5s OF PAGE, WITH  H2-4 OFFSET. FNs MATCH   *
.*     COLUMN, RC ARE IN GUTTER TO ALIGN WITH BINDING.                *
...off
.dh 0 left
.dh 1 left
.dh 2 sect
.dh 3 sect
.dh 4 sect
.fv top
.se *cd = &DH'&$LL / 5
.se *cl = &DH'&$LL - &*cd
.se *rc = &*cd + &DH'2
.se *rc = &*rc.dh
.se *fn = &*cd
.gs args '.rc adjust' '.rc adjust &*rc.' &*fn.dh    0
.gs vars  @rc1         @rc2               @fn1       @fn2
.cd 1 &*cd.dh
.cl &*cl.dh
.rc adjust &*rc
.* DEFINE FN LEADER FOR TWO COLUMN STYLE (OTHERS USE DEFAULT LEADER)  *
...fnldr
.fn leader
.sp 1
.hr @fnldr &@fn1 for &@fnldrlen
.fn off
```

DSM#SUPR

```
.* DSM#SUPR:   Parameter = number(s) to be printed as superscript     *
.***********************************************************************
.go &@suprstyl
.*           FOR 1403 & 3800 OUTPUT WE CAN USE PROPER SUPERSCRIPTS    *
...nums
.tr 0 b0 1 b1 2 b2 3 b3 4 b4 5 b5 6 b6 7 b7 8 b8 9 b9
.ct &*
.tr f0 f0 f1 f1 f2 f2 f3 f3 f4 f4 f5 f5 f6 f6 f7 f7 f8 f8 f9 f9
.if &E'&@fnis ne 0 .is to &@fnis min 1
.me
.*        FOR TERMINALS, THE BEST WE CAN DO IS PUT IT IN PARENTHESES  *
...parens
.if &E'&@fnis eq 0 .ct (&*.)
.th .me
.li on
&*
.li off
.is to &@fnis min 1
.me
.*           FOR PAGE PRINTERS, WE HAVE TO CHANGE FONTS AND SHIFT UP  *
...shifts
.se *hgt1 = &dv'1mv
.bf super =
.se *hgt2 = &dv'1mv
.se *shift = &*hgt1 - &*hgt2
.if &*shift le 0 .th .pf
.th .ct (&*.)
.th .me
.if &E'&@fnis eq 0
.th .sb +&*shift.dv
.th .ct &*.
.th .sb -&*shift.dv
.th .pf
.th .me
.sb +&*shift.dv
.li on
&*
.li off
.sb -&*shift.dv
.pf
.is to &@fnis min 1
```

DSM#TIPG

```
.*  DSM#TIPG:   Creates a title page without rh or rf.             *
.*********************************************************************
.rh sup
.rf sup
.cp
.*  SINGLE COLUMN - NO SPELLCK - NO HYPHENATION                      *
.sa
.ls all 1.0
.sc
.sv off
.hy off
.dc cw
.fo &SYSVART
.sp 2i
.*               SET ARRAY SEPARATOR TO "BREAK"                      *
.'se @ = ';.br;'
.dc asep & @ .
.*    PLACE TITLE IN HIGHLIGHT 2 and DOCUMENT NUMBER, IF ANY         *
.bf title hi2
.ls by &@ttlo
&@title(*)
.ls by 1.0
.pf
.sk 2i
.bf docnum =
.'if &L'&@docnum gt 0 &LL@DocNm &@docnum
.pf
.sk 2
.*                PLACE DOCUMENT DATE                                *
.bf date =
&@docdate
.pf
.sk 1.5i
.*               AUTHOR'S NAMES AND ADDRESSES                        *
.bf author =
&@author(*)
.pf
.sk
.bf address =
&@address(*)
.pf
.sk .5i
.*                 PLACE SECURITY CLASSIFICATION                     *
.bf titlesec =
.li 1
&@sec
.pf
.sk 3
.*           RESTORE FORMATTING ENVIRONMENT & RH/RF. RECLAIM STORAGE  *
.re
.rh res
.rf res
.dm dsm#tipg off
```

DSM#WRIT

```
.*  DSM#WRIT : Write ids out to file.  Define & erase the file first. *
.**********************************************************************
.dm dsm#writ(&$LNUM) /.me
.if &$SYS eq CMS .dd dsmutwtf &SYSVARW DSMREFS
.if &$SYS eq TSO .dd dsmutwtf dsn &SYSVARW..DSMREFS catalog
.if &$SYS ne CMS .an &$SYS ne TSO .me
.wf erase
.*                    LABEL THE FILE APPROPRIATELY                    *
.'wf .* SCRIPT/VS &$DCF.: &LL@device. &$LDEV.
.'wf .* &LL@Refs  &LL@saved. &date. &LL@at. &time.
.su off
.se *parm = substr &$PARM 1  56
.su on
.'wf .* &LL@format.: &*parm
.if &L'&$PARM le 56 .go writout
.se *a = 54
...loop
.su off
.se *parm = substr &$PARM &*a 64
.su on
.'wf .* &*parm
.se *a = &*a + 64
.if &L'&$PARM gt &*a .go loop
...writout
.wf .*
.*         ARRAY @writ@h CONTAINS 1 LINE FOR EACH HEADING "id"        *
.dc asep & @ .
.'se @ = ;
&@writ@h(*)
.*         ARRAY @writ@l CONTAINS 1 LINE FOR EACH FIGURE "id"         *
&@writ@f(*)
.*         ARRAY @writ@d CONTAINS 1 LINE FOR EACH LIST "id"           *
&@writ@d(*)
.*         ARRAY @writ@n CONTAINS 1 LINE FOR EACH FOOTNOTE "id"       *
&@writ@n(*)
.dc asep
```

DSM#WRTD

```
.* DSM#WRTD: Write list ids out to file.                              *
.**********************************************************************
.'wf .'se dl@&*1 = &Dl@&*1
.'wf .'se dP@&*1 = &DP@&*1
```

DSM#WRTF

```
.* DSM#WRTF:  Write figure ids out to file.                           *
.**********************************************************************
.'wf .'se fl@&*1 = &Fl@&*1
.'wf .'se fP@&*1 = &FP@&*1
```

DSM#WRTH

```
.* DSM#WRTH : Write heading ids out to file.                          *
.**********************************************************************
.'wf .'se hl@&*1 '&Hl@&*1
.'wf .'se hP@&*1 '&HP@&*1
```

DSM#WRTN

```
.* DSM#WRTN: Write footnote ids out to file.                          *
.**********************************************************************
.'wf .'se nl@&*1 = &Nl@&*1
```

DSM#XLST

```
.*  DSM#XLST:   No Parms. Internal Service Routine to format cross    *
.*  reference listings. Called by DSMEGDOC or by epifile              *
.*********************************************************************
.dm dsm#xlst(&$LNUM.) /.me
.*          RESET OPEN LISTS, etc.  ADVANCE TO NEXT/ODD PAGE          *
.dsm#rset CrsRf
.dsm#dupl
.*    CLEAR RH/RF, MULTI-COL DEFINITION & PAGINATION FOR CROSS REF    *
.rh sup
.rf sup
.dc cw
.cd 1 0
.cl
.fo left
.bf hi0
.sv off
.*           MACRO @xref@h CONTAINS | LINE FOR EACH HEADING "id"      *
.if &E'&@xref@h eq 0 .go fxref
.dc asep & @ .
.'se @ = ;
&@xref@h(*)
.*           MACRO @xref@f CONTAINS 1 LINE FOR EACH FIGURE "id"       *
...fxref
.if &E'&@xref@f eq 0 .go nxref
.dc asep & @ .
.'se @ = ;
&@xref@f(*)
.*           MACRO @xref@n CONTAINS 1 LINE FOR EACH FOOTNOTE "id"     *
...nxref
.if &E'&@xref@n eq 0 .go ixref
.dc asep & @ .
.'se @ = ;
&@xref@n(*)
.*           MACRO @xref@i CONTAINS 1 LINE FOR EACH INDEX "id"        *
...ixref
.if &E'&@xref@i eq 0 .go lxref
.dc asep & @ .
.'se @ = ;
&@xref@i(*)
.*           MACRO @xref@d CONTAINS 1 LINE FOR EACH LIST "id"         *
...lxref
.if &E'&@xref@d eq 0 .go imtr
.dc asep & @ .
.'se @ = ;
&@xref@d(*)
.* IF &@imtrace EXISTS, FILES WERE IMBEDDED, SO A TRACE IS FORMATTED  *
...imtr
.if &E'&@imtrace eq 0 .me
.in
.of
.sk 4
.cp 3i
.*                FORMAT HEADING FOR IMBED TRACE                      *
.bf hi2
.bx 1 r
.ce &LL@ImTrc
.bx off
.sp 2
.pf
.*            FORMAT EACH ELEMENT IN IMBED TRACE ARRAY                *
.fo off extend
.tb 12m
.dc asep & @ .
.'se @ = ';.br;'
&@imtrace(*)
```

DSM#XRFD

```
.*  DSM#XRFD:  Parm = List Item Id   Formats cross reference for list *
.*  item 'id's.  Defaults set to '?' and blank.                       *
.*********************************************************************
.if &E'&D1@&*1 eq 0 .se D1@&*1 = ?
.if &E'&DP@&*1 eq 0 .se DP@&*1 = ?
.if &E'&DX@&*1 eq 0 .se DX@&*1.(1) = ''
.*                        FORMAT ENTRY                                *
.dm dsm#xrfd(&$LNUM.) off &$CW..dsm#setx D
.of &@xref@of
&*1.&$TAB.&DF@&*1..&$TAB.&DP@&*1..&$TAB.'&D1@&*1..':&$TAB.&DX@&*1.(*)
```

DSM#XRFF

```
.*   DSM#XRFF:  Parm = Figure "id"    Formats cross reference listing *
.*   entry for a figure 'id'. Set defaults of '?' and blank.          *
.*********************************************************************
.if &E'&F1@&*1 eq 0 .se F1@&*1 = ?
.if &E'&FP@&*1 eq 0 .se FP@&*1 = ?
.if &E'&FX@&*1 eq 0 .se FX@&*1.(1) = ''
.*              CREATE LISTING HEADER & ENTRIES                       *
.dm dsm#xrff(&$LNUM.) off &$CW..dsm#setx F
.of &@xref@of
&*1.&$TAB.&FF@&*1..&$TAB.&FP@&*1..&$TAB.&F1@&*1..:&$TAB.&FX@&*1.(*)
```

DSM#XRFH

```
.*   DSM#XRFH:  Parm = Heading "id"   Formats cross reference for     *
.*   heading 'id's. Se defaults to '?' and blank                      *
.*********************************************************************
.if &E'&H1@&*1 eq 0 .se H1@&*1 = ?
.if &E'&HP@&*1 eq 0 .se HP@&*1 = ?
.*               CREATE CROSS REFERENCE HEADER AND ENTRIES            *
.dm dsm#xrfh(&$LNUM.) off &$CW..dsm#setx H
.se *a = &dh'&@xref@of - &dh'4
.of &*a.dh
&*1.&$TAB.&HF@&*1..&$TAB.&HP@&*1..&$TAB.&H1@&*1
.of &@xref@of
.if &E'&HX@&*1 eq 1 &$TAB.&$TAB.&$TAB.&$TAB.&HX@&*1.(*)
```

DSM#XRFI

```
.*   DSM#XRFI:  Parms = Index entry 'id'  Creates cross reference of *
.*   index 'id's.  Defaults set to '?'.                               *
.*********************************************************************
.dm dsm#xrfi(&$LNUM.) off &$CW..dsm#setx I
.if &E'&I1@&*1 eq 0 .se I1@&*1 = ?
.if &E'&IP@&*1 eq 0 .se IP@&*1 = ?
.*                      FORMAT ENTRY                                  *
.se *a = &dh'&@xref@of - &dh'4
.of &*a.dh
&*1.&$TAB.&IF@&*1..&$TAB.&IP@&*1..&$TAB.(1)&$RB.&I1@&*1
.of &*a.dh
.'if &E'&I2@&*1 eq 1 &$TAB.&$TAB.&$TAB.(2)&$RB.&I2@&*1
.of &*a.dh
.'if &E'&I3@&*1 eq 1 &$TAB.&$TAB.&$TAB.(3)&$RB.&I3@&*1
.of &@xref@of
.if &E'&IX@&*1 eq 1 &$TAB.&$TAB.&$TAB.&$TAB.&IX@&*1.(*)
```

DSM#XRFN

```
.*   DSM#XRFN: Parm = Footnote "id".  Formats cross reference for     *
.*   footnote 'id's.  Defaults set to '?' and blank                   *
.*********************************************************************
.if &E'&N1@&*1 eq 0 .se N1@&*1 = ?
.if &E'&NP@&*1 eq 0 .se NP@&*1 = ?
.if &E'&NX@&*1 eq 0 .se NX@&*1.(1) = ''
.*                           FORMAT ENTRY                             *
.dm dsm#xrfn(&$LNUM.) off &$CW..dsm#setx N
.of &@xref@of
&*1.&$TAB.&NF@&*1..&$TAB.&NP@&*1..&$TAB.&N1@&*1..:&$TAB.&NX@&*1.(*)
```

DSM#YESN

```
.*   DSM#YESN:  Parm = "yes" or "no"   Process various attributes     *
.*   which accept only 'yes' or 'no'. Returns answer in &*yesno       *
.*********************************************************************
.se @      = index '-YES-NO-' '-&U'&*1.'
.if &@ eq 0 .se @ = 1
.me .se *yesno = substr 'yes no' &@ 3
```

DSM@FRME

```
.*  DSM@FRME:  Parm = BOX, NONE, RULE   Process FRAME attribute on    *
.*  FIG. Sets &@figframe to type of frame.                            *
.*********************************************************************
.se *a = substr &U'&*1 1 1
.se @figframe '&*
.se @figcw    = '.sx //&@figframe.//
.if &*a eq N .se @figframe off
.if &*a eq R .se @figframe = rule
.th .se @figcw = '.hr @figrule left to right
.if &*a eq B .se @figframe = box
.an &@fig@in eq 0 .se @fig@in = 2
```

DSM@IDS

```
.*   DSM@IDS:  Parm = 'id's  Process all the ID attributes.  Saves     *
.*   file name, id, page # (and figure #, if applicable.) Creates      *
.*   cross reference entries. 'id's are collected only on first pass. *
.*   @tg has been set by the caller to either 'h','f','n','d' or 'i'  *
.**********************************************************************
.if &$PASS ne 1 .dm dsm@ids /.me
.se @TG = &U'&@tg.
.*                  TRUNCATE "ID"S OVER 7 CHARACTERS                   *
.if &L'&*1 le 7 .se *id '&*1
.el .se *id = substr '&*1.' 1 7
.el .dsm#msg 8 &@TG &*1
.*                 IF &&@TG.1. EXISTS, THE "id" IS A DUPLICATE         *
.if &E'&&@TG.1@&*id eq 1 .dsm#msg 9 &@TG &*1
.th .me
.*                  SET UP CROSS REFERENCE ARRAY &@xref@&@tg.          *
.if &SYSVARX ne yes .an &E'&SYSVARW eq 0
.th .go skip
.se @xref@&@tg.() '.dsm#xrf&@tg. &*id
.se @writ@&@tg.() '.dsm#wrt&@tg. &*id
.if &E'&&@TG.F@&*id eq 1 .se *a = &@xref@&@tg.(0)
.th .se @xref@&@tg.(&&@TG.L@&*id..) off
.th .se @xref@&@tg.(0) = &*a
.se &@TG.F@&*id = &$FNAM
.*              SAVE TEXT FOR CROSS REFERENCE PURPOSES                 *
...skip
.se &@TG.P@&*id = &
.if &@TG eq H .se H1@&*id '&@head
.th .me
.if &@TG eq N .se N1@&*id = &@fn#
.th .me
.if &@TG eq F .se F1@&*id = &@fig#
.th .me
.if &@TG eq D .se D1@&*id = '&@id@1.
.th .me
.go index&@ilevel
.*                SET UP INDEX TERMS AND PAGE NUMBERS                  *
...index3 .'se I2@&*id '&#it2
...index2 .'se I1@&*id '&#it1
...index1 .'se I&@ilevel.@&*id '&@it&@ilevel
```

DSM@IPRT

```
.*   DSM@IPRT:  Parm = index term string  Process PRINT attribute on  *
.*   index header tags. Copies value to appropriate &#it.... symbol   *
.**********************************************************************
.'se #it&@ilevel '&*
.*                CONSTRUCT A .PI SORT KEY PARAMETER FOR THE TAG       *
.go key&@ilevel
...key1 .'me .'se *k 'key &X'01.&U'&@it1.&X'010101
...key2 .'me .'se *k 'key &X'0101.&U'&@it2.&X'0101
...key3 .'me .'se *k 'key &X'010101.&U'&@it3.&X'01
```

DSM@MAC@

```
DSMGML3
```

DSM@PGRF

```
.*   DSM@PGRF:  Parm = START, END, MAJOR, NONE  Processes PAGEREF at- *
.*   tribute for index tags. Set's caller's local symbol &*x.         *
.**********************************************************************
.se @ = index      '-START-BEGIN-MAJOR-END-' '-&U'&*1.'
.'if &@ eq 0 .'me .'se *t4 '&*
.me .se *x = substr 'start start order end' &@ 5
```

DSM@PLCE

```
.*   DSM@PLCE  Parm = BOTTOM, COLUMN, INLINE, PAGE, TOP   Processes   *
.*   the PLACE attribute of the FIG tag.                              *
.**********************************************************************
.se @ =  index '-TOP----BOTTOM-INLINE-' '-&U'&*1.'
.if &@ eq 0 .me
.se *b = substr 'top    bottom inline' &@ 6
.se *a = substr '@place @place @place' &@ 6
.se &*a = &*b
```

DSM@PROC

```
.*  DSM@PROC: Parms = a list of process names   Processes the PROCESS *
.*  attribute of the PSC tag.                                         *
.**********************************************************************
.se *i = 1
...loop
.se @ = index '-&$LDEV.-&$PDEV.-&SYSVARP.-' '-&U'&*&*i..-'
.if &@ gt 0 .me
.se *i = &*i + 1
.if &*i le &*0 .go loop
.cs 9 ignore
```

DSM@RFID

```
.* DSM@RFID:  Parameter = "id"  Processed the REFID attribute of the  *
.* cross reference tags.  Sets caller's local symbol *id to value     *
.**********************************************************************
.me .se *id '&*1
```

DSM@RIDI

```
.*   DSM@RIDI:  Parm = index-id  Processes the REFID attribute of the *
.*   I2 and I3 tags.  Provides first and second level index terms.    *
.**********************************************************************
.*                     TRUNCATE 'id's AT 7 CHARACTERS                 *
.if &L'&*1 gt 7 .dsm#msg 8 I &*id
.th .se *id = substr '&*1.' 1 7
.el .se *id = &*1
.*               IF &I1@.... EXISTS, WE CAN GENERATE AN INDEX ENTRY   *
.if &E'&I1@&*id eq 0 .go unknown
.*                        SET #i1 and #i2 TO SAVED TERMS              *
.'se #it1 '&I1@&*id
.'se #it2 '&I2@&*id
.*               SAVE PAGE # FOR CROSS REFERENCING                    *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.se IX@&*id.() = &
.me
.*         SAVE CROSS REFERENCE EVEN IF NOT KNOWN                     *
...unknown
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.if &E'&IF@&*id eq 0 .se @xref@i() '.dsm#xrfi &*id
.th .se IL@&*id = &@xref@i(0)
.th .se IF@&*id = ?
.se IX@&*id.() = &
```

DSM@SEC

```
.* DSM@SEC:  PARM = any string.  Saves security classification        *
.**********************************************************************
.'se @sec '&*
```

DSM@SEE

```
.*   DSM@SEE:  Parms = index reference   Processes SEE attribute of   *
.*   :I1-3 and IREF tags. Assigns  value to caller's local symbol &*r *
.**********************************************************************
.'me .'se *r '&*
```

DSM@SEEI

```
.* DSM@SEEI:  Parm = index-id  Processes SEEID attribute of the :I1-3 *
.* & IREF tags. The index term is returned in the caller's symbol &*r.*
.**********************************************************************
.if &L'&*1 gt 7 .dsm#msg 8 I &*1
.th .se *id = substr '&*1.' 1 7
.el .se *id = &*1
.*          IF &I1@.... EXISTS, WE CAN GENERATE AN INDEX              *
.if &E'&I1@&*id eq 0 .go unknown
.*         CREATE AN INDEX REFERENCE FROM THE SAVED INDEX TERMS.      *
.'se *r '&I1@&*id
.'if &E'&I2@&*id eq 1 .'se *r '&I1@&*id.., &I2@&*id
.'if &E'&I3@&*id eq 1 .'se *r '&I1@&*id.., &I2@&*id.., &I3@&*id
.*         SAVE PAGE # IN &IX @...                                    *
.if &$PASS ne 1 .or &SYSVARX ne yes .go exit
.se IX@&*id.() = &
.go exit
.*      IF "id" IS UNKNOWN, RETURN A "?" AS THE REFERENCED TEXT       *
...unknown
.if &$PASS ne 1 .or &SYSVARX ne yes .me .se *r = ?
.*                 REMEMBER 'ID' FOR CROSS REFERENCING                *
.se *r = ?
.if &E'&IF@&*id eq 0 .se @xref@i() '.dsm#xrfi &*id
.th .se IL@&*id = &@xref@i(0)
.th .se IF@&*id = ?
.se IX@&*id.() = &
.*          RETURN REFERENCE TEXT IN CALLER'S LOCAL SYMBOL &*r        *
...exit
.'me .'se *r '&*r
```

DSM@SHD

```
.*  DSM@SHD:  Parm = any string of text  Processes STITLE attribute   *
.*  for the H1 tag.                                                   *
.**********************************************************************
.'se @shead '&*
```

DSM@STTL

```
.*  DSM@STTL:  Parms = Any string of text   Processes the STITLE      *
.*  attribute of the TITLE tag.                                       *
.**********************************************************************
.se @stinit = 1
.'se @stitle '&*
```

DSM@WIDT

```
.* DSM@WIDT:  Parms = COLUMN, PAGE, space-unit. Processes the WIDTH   *
.* attribute of the FIG tag. Sets &@place to "column" or "page".      *
.**********************************************************************
.se @ =  index                   '-PAGE---COLUMN-' '-&U'&*.'
.if &@ ne 0 .me .se @width = substr 'page   column' &@ 6
.*        COMPARE SPACE AMOUNT TO COLUMN LINE LENGTH                  *
.if &DH'&*1 gt &DH'&$CL .se @width = page
.el .se @width = column
```

DSMABSTR

```
.* DSMABSTR: Tag = ABSTRACT  No  Attributes   Sets up formatting      *
.* environment for abstract.  Generates and h1.  Resets open lists,   *
.* etc..  Advances to next/odd page. Sets &@shead for running footing *
.**********************************************************************
.dsm#rset Abstr
.dsm#dupl
.'se @shead '&LL@Abstr
.'h1 &LL@Abstr
.*            FIRST PARAGRAPH IN ABSTRACT SHOULDN'T BE INDENTED       *
.aa p dsmpara1
```

DSMADDR

```
.*  DSMADDR:  Tag = ADDRESS  No Attributes  In title page, saves the  *
.*  lines of the address.  Off title page, formats address as simple  *
.*  list. Enables ALINE tag and gets the residual text.               *
.*  &@addctr counts # of addresses & is used to construct the name of *
.*  the array containing the address lines. &@address contains the    *
.*  name of each address line array (one address array per element).  *
.*********************************************************************
.aa aline dsmaline
.gs scan *line
.if &@state ne TtlPg .go inline
.*           SET UP @addctr and @address ARRAY                         *
.se @addctr = &@addctr + 1
.se @aline&@addctr off
.'se @address() = ';.sp;&@aline&@addctr.(*).'
.*                 RESIDUAL TEXT IS 1ST LINE OF ADDRESS                *
.'if &L'&*line gt 0 .'se @aline&@addctr.() '&*line
.me
.*      ADDRESSES NOT ON TITLE PAGE, ARE FORMATTED AS A SIMPLE LIST   *
...inline
.sk &@sk@s
.sa
.fo off
.in +&@in@s
.if &@state eq open .kp on
.if &L'&*line eq 0 .me
.li 1
&*line
```

DSMALINE

```
.* DSMALINE:  Tag = ALINE  No Attributes   On title page, saves line  *
.* in symbol array. Outside a title page, the lines are formatted as  *
.* a compact simple list.                                              *
.*********************************************************************
.if &@state ne TtlPg .me
.gs scan *line
.'se @aline&@addctr.() '&*line
```

DSMAPPD

```
.*  DSMAPPD: Tag = APPENDIX   No Attributes  Set up for appendices     *
.*  H1s in Appendix are preceded w/ 'Appendix' & a serial letter      *
.*********************************************************************
.gs hctr A.0
.se @head1 '&LL@Appdx
.if &SYSVARH ne no .se *a = num
.dh 0 tc
.dh 1 tc nonum
.dh 2 tc &*a
.dh 3 tc &*a
.dh 4 tc &*a
.*          RESET ANY OPEN LISTS, ETC.  ADVANCE TO NEXT/ODD PAGE      *
.dsm#rset Appdx
.dsm#dupl
.*    ESTABLISH SAME PAGE LAYOUT AS BODY & RESET NORMAL PAGE NUMBERS  *
.dsm#styl
.pn arabic
```

DSMAUTHR

```
.*  DSMAUTHOR:  Tag = AUTHOR  No Attributes  Saves text in             *
.*  &@author array.  Valid only on title page.                         *
.*********************************************************************
.gs scan *line
.'se @author() '&*line
```

DSMBACKM

```
.*  DSMBACKM: Tag = BACKM  No Attributes   Sets up for back matter.   *
.*  Resets any open lists, etc.  Headings are in toc but not          *
.*  numbered. Advances to next/off page and sets up 2 column format.  *
.*********************************************************************
.dsm#rset BkMtr
.se @headl off
.dh 0 tc
.dh 1 tc nonum
.dh 2 tc nonum
.dh 3 tc nonum
.dh 4 tc nonum
.dsm#dupl
.dsm#styl two
```

DSMBODY

```
.* DSMBODY: Tag = BODY,No Attributes. Sets up for body section. Head- *
.* ings are in toc and may be numbered.  Resets any open lists, etc.  *
.* Advances to next/odd page, sets up page layout, resets page to '1' *
.*************************************************************@BA34319
.se @shead =
.if &E'&@bodyheadl eq 1 .se @headl '&@bodyheadl
.if &SYSVARH ne no .se *a = num
.dh 0 tc
.dh 1 tc nonum
.dh 2 tc &*a
.dh 3 tc &*a
.dh 4 tc &*a
.an &E'&@headl eq 0 .dh 1 num
.dsm#rset Body
.dsm#dupl
.dsm#styl
.pn arabic
.pn 1
```

DSMCIT

```
.*  DSMCIT: Tag = CIT  No Attributes  Starts highlight for citations  *
.*********************************************************************
.bf hil althil
```

DSMDATE

```
.*  DSMDATE: Tag = DATE  No Attributes  Save text in &@docdate or     *
.*  sets it to the current date.  Valid only in front matter.         *
.*********************************************************************
.gs scan @docdate
.'if &L'&@docdate eq 0 .'se @docdate '&date
.'el .'se date '&@docdate
```

DSMDCNUM

```
.*  DSMDCNUM:  Tag = DOCNUM  No Attributes  Saves number in @docnum   *
.*********************************************************************
.gs scan @docnum
```

DSMDDEF

```
.* DSMDDEF:  Tag = DD   No Attributes    Definition term is in &@id@l *
.* issue msg if not.  Handles any definition headings left around.    *
.**********************************************************************
.if &L'&@id@l eq 0 .dsm#msg 6 Def
.th .se @id@l = ?
.if &E'&@dthead eq 0 .go desc
.se *k = &DV'3mv + &DV'&@sk@l
.kp &*k.dv
.sk &@sk@l c
.in &@in
.bf hi&@hi@hd =
.li 1
&@dthead
.pf
.sk &@sk@l c
.se @dthead off
.*   ESTABLISH FORMATTING ENVIRONMENT FOR DEFINITION LIST ITEMS       *
...desc
.sk &@sk@l c
.in &@in
.in +&@in@l after 1
.*          FORMAT DEFINITION TERM IN THE APPROPRIATE FONT            *
.bf hi&@hi@l =
.li 1
&@id@l.
.pf
.is to &@li@tab min 1 &@break
.*      SET &@id@l TO NULL TO INDICATE A PAIR HAS BEEN PROCESSED      *
.se @id@l = ''
```

DSMDDHD

```
.*  DSMDDHD:  Tag = DDHEAD  No Attr. Text is heading for definition   *
.*  descriptions. If &@dthead is null, there was no heading for terms *
.**********************************************************************
.if &E'&@dthead eq 0 .se @dthead = ''
.se *k = &DV'3mv + &DV'&@sk@l
.kp &*k.dv
.sk &@sk@l c
.in &@in
.in +&@in@l after 1
.*    GET THE DESCRIPTION HEADING & FORMAT IN CORRECT FONT            *
.gs scan *ddhead
.bf hi&@hi@hd =
.li 1
&@dthead.
.is to &@li@tab min 1
.li 1
&*ddhead
.pf
.sk &@sk@l c
.*              SET &@dthead TO NULL TO INDICATE DONE                 *
.se @dthead off
```

DSMDLIST

```
.*  DSMDLIST: Tag = DL No Attributes Calls DSMLISTM to process line   *
.**********************************************************************
.dsmlistm * &*
```

DSMDTERM

```
.* DSMDTERM: Tag = DT  No Attr. Save text as term ro succeeding :DD   *
.* If &@id@l is not null, 2 :DT tags have been found & 1st is ignored *
.**********************************************************************
.if &L'&@id@l ne 0 .dsm#msg 5 Def '&@id@l.'
.*            CHECK FOR UNPRINTED DEFINITION HEADINGS LEFT AROUND     *
.if &E'&@dthead eq 0 .go term
.se *k = &DV'3mv + &DV'&@sk@l
.kp &*k.dv
.sk &@sk@l c
.in &@in
.bf hi&@hi@hd =
.li 1
&@dthead
.pf
.sk &@sk@l c
.se @dthead off
.*                     SAVE RESIDUAL TEXT IN &@id@l.                  *
...term
.sk &@sk@l c
.gs scan @id@l
```

DSMDTHD

```
.*  DSMDTHD:  Tag = DTHEAD  No attr. Saves residual text for          *
.*  succeeding :DDHD tag or :DT tag.                                  *
.**********************************************************************
.sk &@sk@l c
.gs scan @dthead
```

DSMEADDR

```
.*  DSMEADDR:   Tag = ADDRESS end  No attr    Disables the :ALINE     *
.*  tag. If not on a title page, end the simple list with skip        *
.**********************************************************************
.aa aline dsm#cntx
.if &@state eq TtlPg .me
.if &@state eq open .kp off
.re
.sk &@sk@s c
```

DSMECIT

```
.*  DSMECIT:  Tag = CIT end tag  No Attr.  Ends highlighting          *
.**********************************************************************
.pf
```

DSMEFIG

```
.*  DSMEFIG:  Tag = FIG end-tag   No Attr  Restores environment      *
.*********************************************************************
.if &@state ne F .dsm#msg 11 F
.th .me
.se @state = open
.*               CLEAR INDENTION AND FINISH THE FRAME                *
.if &@efigpf ne no .pf
.in
.ir
.bf hi2
.if &@figframe eq box .bx off
.th .go @frdone
.if &@place ne bottom .an &E'&@figframe eq 1 .an /&@figframe ne /box
.th &@figcw
...@frdone
.pf
.*            END KEEP OR FLOAT & RESTORE SAVED ENVIRONMENT          *
.if &@place eq top .sp &@sk@f
.&@figtype off
.if &@place eq inline .sp &@sk@f c
.re
.*          :FIGCAP & :FIGDESC TAGS ARE INVALID OUTSIDE A FIGURE     *
.aa figcap dsm#cntx
.aa figdesc dsm#cntx
```

DSMEFTNT

```
.*  DSMEFTNT:    Tag = FN end-tag   No attr.  ENDS FOOTNOTE          *
.*********************************************************************
.if &@state ne N .dsm#msg 11 N
.th .me
.se @state = open
.pf
.fn off
```

DSMEGDOC

```
.*  DSMEGDOC: Tag = EGDOC  No Attr  Produces cross reference listing  *
.*********************************************************************
.if &SYSVARX eq yes .an &@lastpass eq yes .dsm#xlst
.if &E'&SYSVARW ne 0 .an &@lastpass eq yes .dsm#writ
```

DSMEHP

```
.*  DSMEHP:  tag = HP end-tag   No Attr ENDS HIGLIGHTING             *
.*********************************************************************
.pf
```

DSMELIST

```
.*  DSMELIST: Tag = list end-tag  No Attr. Restores formatting para-  *
.*  meters saved when the list was started. &@nest@l = current level  *
.*  of nesting.  See Starter Set Implementation Guide for details.    *
.*********************************************************************
.if &@nest@l(0) eq 0 .dsm#msg 11 List
.th .me
.*            NESTING LEVEL 1 - RESTORE ENVIRONMENT TO OPEN TEXT       *
.if &@nest@l(0) gt 1 .go denest
.se @sk@l = &@sk@ls
.se @hi@hd = &@hi@h
.sk &@sk@l c
.in &@in
.se @nest@l(0) = 0
.se @nest@o off
.se @nest@u off
.aa li dsm#cntx
.aa lp dsm#cntx
.aa dt dsm#cntx
.aa dd dsm#cntx
.aa dthd dsm#cntx
.aa ddhd dsm#cntx
.aa gt dsm#cntx
.aa gd dsm#cntx
.me
.*              DECREMENT APPROPRIATE LIST NESTING COUNTER            *
...denest
.if &E'&@nest@&@ltype eq 1
.th .se @nest@&@ltype = substr &@denest@&@ltype &@nest@&@ltype 1
.sk &@sk@l c
.*              RESTORE FORMATTING PARAMETERS FROM PREVIOUS LIST      *
.gs args &V'&@nest@l(&@nest@l(0).)
.gs vars @ltype @item# @in @in@l @sk@l @hi@l @hi@hd @break
.se @nest@l(0) = &@nest@l(0) - 1
.in &@in
.in +&@in@l
.se @li@tab = &DH'&$IN.dh
.if &@ltype eq u .or &@ltype eq o
.th .se *a = substr &@&@ltype.listnest &@nest@&@ltype 1
.se @id@l '&V'&@id@l@&@ltype.&*a..'
.*              REENABLE THE APPROPRIATE TAGS                         *
.aa li dsmlitem
.if &@ltype ne d
.an &@ltype ne g .th .me
.aa li dsm#cntx
.*
.if &@ltype eq d
.th .aa dt dsmdterm
.th .aa dd dsmddef
.th .aa dthd dsmdthd
.th .aa ddhd dsmddhd
.th .aa gt dsm#cntx
.th .aa gd dsm#cntx
.th .me
.*
.if &@ltype eq g
.th .aa gt dsmgterm
.th .aa gd dsmgdef
.th .aa dt dsm#cntx
.th .aa dd dsm#cntx
.th .aa dthd dsm#cntx
.th .aa ddhd dsm#cntx
```

DSMELQU

```
.*  DSMELQU:  Tag = LQ end-tag   No Attr.  ENDS LONG QUOTES          *
.*********************************************************************
.pf
.sk &@sk@q c
.ir -&@in@q
.*             &@nest@l INDICATES CURRENT NESTING LEVEL              *
.if &@nest@l(0) eq 0 .dsm#msg 11 List
.th .me
.*             RESTORE OPEN TEXT ENVIRONMENT IF NESTING LEVEL IS 1   *
.if &@nest@l(0) gt 1 .go denest
.se @nest@l(0) = 0
.in &@in
.me
.*             RESTORE PRIOR LIST'S PARAMETERS FROM &@nest@l         *
...denest
.gs args &V'&@nest@l(&@nest@l(0).)
.gs vars @ltype @item# @in @in@l @sk@l @hi@l @hi@hd @break
.se @nest@l(0) = &@nest@l(0) - 1
.if &@ltype eq u .or &@ltype eq o
.th .se *a = substr &@&@ltype.listnest &@nest@&@ltype 1
.se @id@l '&V'&@id@l@&@ltype.&*a..'
.*        REESTABLISH FORMATTING ENVIRONMENT FOR PRIOR LIST TYPE    *
.in &@in
.in +&@in@l
.se @li@tab = &DH'&$IN.dh
```

DSMEPSC

```
.*  DSMEPSC:  Tag = PSC end-tag  No Attr.  Ends condition section    *
.*********************************************************************
.cs 9 off
```

DSMEQUOT

```
.*  DSMEQUOT:  Tag = QUOTE end tag  No Attr.  Ends short quotes.     *
.*********************************************************************
.if &@nest@q eq 0 .dsm#msg 11 QtePh
.th .me
.su off
.*             &@nest@q INDICATES THE CURRENT NESTING LEVEL          *
.*             EXTRACT THE CORRECT QUOTATION DELIMITER FROM &@cquote *
.se *q = substr &@cquote &@nest@q 1
.se @nest@q = &@nest@q - 1
.su on
.ct &*q
```

DSMETTLP

```
.*  DSMETTLP: Tag = ETITLEP  No Attr. Calls DSM#TIPG to create       *
.*  title page                                                       *
.*********************************************************************
.if &@state ne TtlPg .dsm#msg 11 TtlPg
.th .me
.se @state = open
.if &SYSVART ne no .dsm#tipg
.*              TURN OFF TAGS NOT ALLOWED OFF OF TITLE PAGE          *
.aa author  dsm#cntx
.aa date    dsm#cntx
.aa docnum  dsm#cntx
.aa title   dsm#cntx
```

DSMEXMP

```
.*  DSMEXMP:  Tag = XMP end-tag  No Attr. End the keep, restore for-   *
.*  matting environment . Reset &@nest@x to indicate example ended.    *
.*********************************************************************
.if &@state ne Exmpl .dsm#msg 11 Exmpl
.th .me
.se @state = open
.kp off
.pf
.re
.sk &@sk@x c
```

DSMFCAP

```
.*  DSMFCAP:  Tag = FIGCAP   No Attr.Formats text w/ figure #. Puts    *
.*  entries into #FIGLIST  macro for list of illustrations.            *
.****************************************************************@BA34232
.*    ESTABLISH FORMATTING ENVIRONMENT FOR CAPTION AND DESCRIPTION     *
.pf
.se @efigpf = no
.gs scan *line
.in &@fig@in
.ir &@fig@in
.fo on
.sv on
.sp 1 c
.*     PENDING INDENTION TO ALIGN FIGURE DESCRIPTION WITH CAPTION      *
.bf figcap =
.se *w = 3 * &DH'&W'0
.se *period '.
.se *a = &DH'&W'&@fig# + &DH'&W'&LL@F + &*w + &DH'&W'&*period
.se *b = &*a + &DH'&$IN
.in &*b.dh
.un &*a.dh
.*         FORMAT FIGURE CAPTION  PREFIXED WITH FIGURE NUMBER          *
&LL@F.&$RB.&@fig#..
.is to &*b.dh min 1
&*line.
.pf
.*           MAKE AN ENTRY FOR #FIGLIST                                *
.if &@fig# lt 10 .se *pad = &$RB
.'se *sx '&X'00.&LL@F.&$RB.&*pad.&@fig#..&$RB.&*line.&X'00. .&X'00
.'dm #figlist( ) &X'01..of &L'XXXXX&LL@F ;.'sx f &*sx.&@FN#&@fig#
.se @FN#&@fig# = &
.*          INCREMENT FIGURE COUNTER AND INDICATE CAPTION WAS FOUND    *
.se @fig# = &@fig# + 1
.se @fig@fo = on
```

DSMFDESC

```
.*  DSMFDESC: Tag = FIGDESC  No Attr. Appends figure description to a *
.*  figure caption.  Prepares environment only if there's no caption   *
.*****************************************************************@BA34232
.if &@fig@fo eq off .go format
.se @fig@fo = off
.bf figcap =
.ct :&$RB.&$RB.&$CONT
.pf
.bf figdesc =
.me
.*           NO FIGURE CAPTION - ESTABLISH ENVIRONMENT                 *
...format
.pf
.se @efigpf = no
.sp
.in &@fig@in
.ir &@fig@in
.fo on
.sv on
.bf figdesc =
```

DSMFGREF

```
.*  DSMFGREF:  Tag = FIGREF  Attr = PAGE, REFID  Inserts text of       *
.*  figure references. Includes figure number, and may include PAGE    *
.*  ref &*yesno will be "no" if PAGE attribute was given.              *
.*  The REFID attribute will set &*id.                                 *
.***********************************************************************
.gs exatt refid as dsm@rfid page as dsm#yesn
.if &L'&*id gt 7 .dsm#msg 8 F &*id
.th .se *id = substr '&*id.' 1 7
.*         ID &Fl@.... EXISTS, WE CAN GENERATE A STRING W/ FIGURE #    *
.se *F='F'
.if &E'&Fl@&*id eq 0 .an &E'&fl@&*id eq 0 .go unknown
.if &E'&Fl@&*id eq 0 .se *F = f
.se *p = &
.*              IF PAGE NOT SUPPRESS, GENERATE PAGE REFERENCE          *
.if /&*yesno eq /yes .or &*p ne &&*F.P@&*id .an /&*yesno ne /no
.th .se *r ' &LL@onpge &&*F.P@&*id
&LL@F.&$RB.&&*F.l@&*id..&*r.&$CONT
.*              GENERATE CROSS REFERENCE PAGE NUMBER IN &FX@...        *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.se FX@&*id.() = &
.me
.*             IF &Fl@.... DOES NOT EXIST YET, USE CANNED STRING       *
...unknown
.sv off
-- &LL@F id '&*id.' &LL@unkn --&$CONT
.sv on
.*           GENERATE THE CROSS REFERENCE INFORMATION                  *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.if &E'&FF@&*id eq 0 .se @xref@f() '.dsm#xrff &*id
.th .se FL@&*id = &@xref@f(0)
.th .se FF@&*id = ?
.se FX@&*id.() = &
```

DSMFIG

```
.*  DSMFIG: Tag = FIG  Attr = DEPTH, FRAME, ID, PLACE  Start a        *
.*  figure. Generates a keep or a float, depending on PLACE attri-    *
.*  bute.  Generates a box or rule depending on FRAME attribute.      *
.*********************************************************************
.if &@state ne open .dsm#msg 4 &$TAG &@state
.th .me
.se @state = F
.*               SAVE CURRENT FORMATTING ENVIRONMENT                  *
.br
.sa
.*     ESTABLISH DEFAULT FRAME AND TYPE AND PROCESS THE ATTRIBUTES    *
.gs args rule      &@in@f  off     fl       &@figplace &@figwidth off
.gs vars @figframe @fig@in @fig@fo @figtype @place     @width   @efigpf
.se @figcw = '.hr @figrule left to right
.gs exatt frame as dsm@frme width as dsm@widt place as dsm@plce
.*         ESTABLISH FORMATTING ENVIRONMENT FOR FIGURE                *
.in
.ir
.fo off
.sv off
.*                   BEGIN KEEP OR FLOAT                              *
.*
.if &@place eq inline .se @figtype = kp
.an &@width eq page .sc
.if &@place eq inline .sp &@sk@f
.&@figtype on &@place &@width order
.if &@place eq bottom .sp &@sk@f
.*                  START FRAME                                       *
.ls all 1.0
.ws
.es
.bf hi2
.if &@figframe eq box .bx @figrule new left right
.th .go @frdone
.if &@place ne top .an &E'&@figframe eq 1 .an /&@figframe. ne /box
.th &@figcw
...@frdone
.pf
.*                   ESTABLISH PROPER INDENTION FOR FIGURES           *
.in &@fig@in
.ir &@fig@in
.*        PROCESS ID AND DEPTH ATTRIBUTES AND ENABLE APPROPRIATE TAGS  *
.se @tg = f
.gs exatt id as dsm@ids depth as sp
.aa figcap dsmfcap
.aa figdesc dsmfdesc
.bf figfont =
.se @efigpf = yes
```

DSMFLIST

```
.*  DSMFLIST:  Tag = FIGLIST   No Attr.  Formats list of illustra-    *
.*  tions.  Resets any open lists, etc.  Advances to next/odd page.   *
.*  Sets @shead for the running footing.  Puts an h1 into DSM#FLIST.  *
.*  The DSM#FLIST macro has been built by the FIGCAP tag.             *
.*********************************************************************
.dsm#rset LstIl
.dsm#dupl
.'se @shead '&LL@LstIl
.dm #figlist(1) /.'h1 &LL@LstIl
.sa
.dc cw
.#figlist
.re
.pa nostart
```

DSMFNREF

```
.*  DSMFNREF:  Tag = FNREF  Attr = REFID   GENERATE FN CALL-OUT       *
.*********************************************************************
.gs exatt refid as dsm@rfid
.if &L'&*id gt 7 .dsm#msg 8 F &*id
.th .se *id = substr '&*id.' 1 7
.*          IF &N1@.... EXISTS, CALL DSM#SUPR TO INSERT SUPERSCRIPT  *
.if &E'&N1@&*id eq 1 .dsm#supr &N1@&*id
.if &E'&n1@&*id eq 0 .an &E'&N1@&*id eq 0 .dsm#supr 00
.if &E'&n1@&*id eq 1 .an &E'&N1@&*id eq 0 .dsm#supr &n1@&*id
.*          IF CROSS REFERENCING, SAVE CURRENT PAGE # IN &NX@...     *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.if &E'&N1@&*id eq 0 .go unknown
.se NX@&*id.( ) = &
.me
.*             GENERATE CROSS REFERENCE INFORMATION                  *
...unknown
.if &E'&NF@&*id eq 0 .se @xref@n( ) '.dsm#xrfn &*id
.th .se NL@&*id = &@xref@n(0)
.th .se NF@&*id = ?
.se NX@&*id.( ) = &
```

DSMFRONT

```
.*  DSMFRONT: Tag = FRONTM  No Attr.  Establishes formatting         *
.*  environment for front matter section. Heading don't go into      *
.*  toc and are not numbered                                         *
.*********************************************************************
.se @headl off
.dh 0 ntc
.dh 1 ntc nonum
.dh 2 ntc nonum
.dh 3 ntc nonum
.dh 4 ntc nonum
.*   ADVANCE TO THE NEXT/ODD PAGE. ESTABLISH OFFSET OR ONE COL SYTLE  *
.dsm#dupl
.if &SYSVARS eq two .dsm#styl one
.el .dsm#styl
.pn roman
```

DSMFTNT

```
.*  DSMFTNT: Tag = FN  Attr = ID  Starts a footnote.  Generates       *
.*  footnote call-out, if no ID given.                                *
.*********************************************************************
.if &@state ne open .dsm#msg 4 &$TAG &@state
.th .me
.se @state = N
.*                NO ID - ASSUME CALL-OUT GOES RIGHT HERE             *
.gs qatt *q id
.if &E'&*q ne 1 .go strtfn
.dsm#supr &@fn#
.*                      START FOOTNOTE - PROCESS ID                   *
...strtfn
.fn on
.bf fnt =
.sp &@sk@n
.in &@fnl
.se @tg = n
.if &E'&*q eq 0 .gs exatt id as dsm@ids
.if &@suprstyl eq shifts .bf super =
.se @fnis = &DH'&W'0 * 4 + &DH'&$IN
.se @fnis = '&@fnis.dh
.ir &@fn2
.se *i = &DH'&W'0 * 4
.if &@suprstyl eq shifts .pf
.in +&*i.dh after 1
.dsm#supr &@fn#
.se @fn# = &@fn# + 1
.se @fnis off
```

DSMGDEF

```
.*  DSMGDEF:  Tag = GD   No Attr.  Processes glossary definition.     *
.*  The glossary term is in &@id@l.                                   *
.*********************************************************************
.if &L'&@id@l eq 0 .dsm#msg 6 Gloss
.th .se @id@l = ?
.*                 FORMAT THE GLOSSARY TERM IN A BOLD FONT            *
.sk &@sk@l
.in &@in
.bf hi&@hi@l =
.li 1
&@id@l.:&$CONT
.pf
&$RB.
.*                 SET &@id@l TO NULL TO INDICATE WE DID IT           *
.se @id@l = ''
```

DSMGDOC

```
.*  DSMGDOC: Tag = GDOC  Attr = SEC  Processes SEC attribute          *
.*********************************************************************
.gs exatt sec as dsm@sec
.dm dsmgdoc off
```

DSMGLIST

```
.*  DSMGLIST:  Tag = GL  No Attr. Calls list macro to start glossary  *
.*********************************************************************
.dsmlistm g &*
```

DSMGTERM

```
.*  DSMGTERM:  Tag = GT  No Attr. Saves glossary term in &@id@l.      *
.*********************************************************************
.if &L'&@id@l ne 0 .dsm#msg 5 Gloss '&@id@l.'
.sk &@sk@l c
.gs scan @id@l
```

DSMHDREF

```
.*  DSMHDREF:  Tag = HDREF  Attr = PAGE, REFID  Inserts heading cross *
.*  reference text (heading and perhaps page number) into document    *
.*  If PAGE was given &*yesno will be 'no'.  REFID will set &*id.     *
.*********************************************************************
.gs exatt refid as dsm@rfid page as dsm#yesn
.if &L'&*id gt 7 .dsm#msg 8 H &*id
.th .se *id = substr '&*id.' 1 7
.*         IF &H1@.... EXISTS, WE CAN GENERATE A CROSS REFERENCE      *
.se *H ='H'
.if &E'&H1@&*id eq 0 .an &E'&h1@&*id eq 0 .go unknown
.if &E'&H1@&*id eq 0 .se *H = h
.su off
.se *cw = &$CW
.dc cw off
.se *p = &
.se *n = &@nest@q + 1
.se *o = substr &@oquote &*n 1
.se *c = substr &@cquote &*n 1
.*          DECIDE WHETHER OR NOT TO GENERATE PAGE REF               *
.if &*yesno eq yes .or &*p ne &&*H.P@&*id .an &*yesno ne no
.th .se *r ' &LL@onpge &&*H.P@&*id..'
.*              GENERATE THE TEXT OF THE HEADING REFERENCE            *
.su on
&*o.&&*H.1@&*id..&*c.&*r.&$CONT
.dc cw &*cw
.*            SAVE CURRENT PAGE NUMBER IN &HX@...
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.se HX@&*id.() = &
.me
.*            IF &H11@.... DOES NOT EXIST, USE A CANNED STRING        *
...unknown
.sv off
-- &LL@H id '&*id.' &LL@unkn --&$CONT
.sv on
.*           SAVE CROSS REFERENCE INFORMATION                         *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.if &E'&HF@&*id eq 0 .se @xref@h() '.dsm#xrfh &*id
.th .se HL@&*id = &@xref@h(0)
.th .se HF@&*id = ?
.se HX@&*id.() = &
```

DSMHEAD0

```
.*  DSMHEAD0: Tag = H0  ATTR. = ID, STITLE  Formats head 0 using res- *
.*  idual text.  Resets open lists, etc, advances to next/odd page    *
.*********************************************************************
.dsm#rset H.-0
.dsm#dupl
.gs scan @head
.*            SET &@shead FOR RUNNING FOOTING TO HEADING OR STITLE    *
.'se @shead '&@head
.gs exatt stitle as dsm@shd
.*      CREATE HEAD 0, PROCESS ID ATTRIBUTE & DON'T INDENT PARAGRAPH  *
.'h0 &@head
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara1
```

DSMHEAD1

```
.*  DSMHEAD1: Tag = H1  Attr = ID, STITLE  Format level 1 heading.    *
.*  Advances to next/odd page.  Head1's are numbered in the body if   *
.*  either head level numbering is on or &@head1 exists.              *
.*  Reset any open lists, etc.                                        *
.**********************************************************************
.dsm#rset H.-1
.dsm#dupl
.gs scan @head
.*          PREFIX HEADING WITH &@head1, IF IT EXISTS                 *
.if &E'&@head1 eq 1 .gs hctr 1
.'th .'se @head '&@head1 &@xref(1).. &@head
.*         SET &@shead FOR THE RUNNING FOOTING TO HEADING OR STITLE  *
.'se @shead '&@head
.gs exatt stitle as dsm@shd
.* CREATE THE HEADING, PROCESS THE ID AND DON'T INDENT 1ST PARAGRAPH  *
.'h1 &@head
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara1
```

DSMHEAD2

```
.* DSMHEAD2: Tag = H2   Attr = ID  Formats level 2 heading            *
.* Resets any open lists, etc.                                        *
.**********************************************************************
.dsm#rset H.-2
.gs scan @head
.* CREATE THE LEVEL 2 HEADING, PROCESS THE ID & DON'T INDENT 1ST PARA *
&@rc1
.'h2 &@head
&@rc2
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara2
```

DSMHEAD3

```
.* DSMHEAD3: Tag = H3         Attributes = ID                         *
.* Formats a level three heading.  Resets any open lists, etc.        *
.**********************************************************************
.dsm#rset H.-3
.gs scan @head
.* CREATE THE LEVEL 3 HEADING, PROCESS THE ID, DON'T INDENT 1ST PARA  *
&@rc1
.'h3 &@head
&@rc2
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara2
```

DSMHEAD4

```
.*  DSMHEAD4:   Tag = H4       Attributes = ID                        *
.*  Formats a level four heading. Resets any open lists, etc.         *
.**********************************************************************
.dsm#rset H.-4
.gs scan @head
.* CREATE THE LEVEL 4 HEADING, PROCESS THE ID, DON'T INDENT 1st PARA  *
&@rc1
.'h4 &@head
&@rc2
.se @tg = h
.gs exatt id as dsm@ids
.aa p dsmpara2
```

DSMHEAD5

```
.*  DSMHEAD5:  Tag = H5       Attributes = ID                         *
.*  Formats a level five heading in-line. Resets any open lists, etc. *
.**********************************************************************
.dsm#rset H.-5
.gs scan @head
.*                CREATE THE LEVEL 5 HEADING, PROCESS THE ID          *
.'h5 &@head
.se @tg = h
.gs exatt id as dsm@ids
.*                THE 1st :P TAG WILL ADD A COLON TO THE HEADING      *
.se @h5line = '&$PN./&$LC.'
.se @para5@fnt = 5
.aa p dsmpara5
```

DSMHEAD6

```
.*  DSMHEAD6:   Tag = H6       Attributes = ID                        *
.*  Formats a level six heading in-line. Resets any open lists, etc.  *
.**********************************************************************
.dsm#rset H.-6
.gs scan @head
.*                     CREATE THE LEVEL 6 HEADING, AND PROCESS THE ID  *
.'h6 &@head
.se @tg = h
.gs exatt id as dsm@ids
.*                       THE 1st :P TAG WILL ADD A COLON              *
.se @h5line = '&$PN./&$LC.'
.se @para5@fnt = 6
.aa p dsmpara5
```

DSMHP0

```
.*  DSMHP0:   Tag = HP0   No Attr.  Starts the body font              *
.**********************************************************************
.bf hi0
```

DSMHP1

```
.*  DSMHP1:  Tag = HP1   No Attr.   Start level 1 highlighting        *
.**********************************************************************
.bf hi1 althi1
```

DSMHP2

```
.*  DSMHP2:  Tag = HP2  No Attr   Start level 2 highlighting          *
.**********************************************************************
.bf hi2 althi2
```

DSMHP3

```
.*  DSMHP3:  Tag = HP3   No Attr   Start level 3 highlighting.        *
.**********************************************************************
.bf hi3 althi3
```

DSMIDMMY

```
.*  DSMIDMMY: Parms = index term.  Processes index tags when no       *
.*  index is being produced.   Consumes residual text.                *
.**********************************************************************
.gs scan *x
```

DSMIEH

```
.*  DSMIEH:   Formats index headers .                                    *
.*********************************************************************
.if /&*1 lt /a .me
.*                       INSERT WHITE SPACE, START A KEEP
.sk p22
.if &$ENV eq KP .kp off
.kp 1.2i
.bf ieh hi2 althi2
.*         PRINT THE PARAMETER IN A BOLD FONT, SURROUNDED BY A BOX      *
.in 5
.se *a = 7 + &L'&*
.if SYSOUT ne PAGE .bx 4 &*a
.el .in 0
.li 1
&*
.if SYSOUT ne PAGE .bx off
.sk 1
.pf
.in
```

DSMIHD1

```
.*  DSMIHD1:   Tag = IH1       Attributes = ID, PRINT, SEE, SEEID        *
.*  Creates an index header  (an index entry w/ no page numbers)         *
.*  Get residual text into &@it1 and copy it to &#it1.                  *
.*********************************************************************
.gs scan @it1
.'se #it1 '&@it1
.*               &@it2 AND &@it3 ARE NULL FOR LEVEL ONE TERMS.         *
.gs args 1       ''   ''
.gs vars @ilevel @it2 @it3
.* THE PRINT ATTRIBUTE MAY RESET &@#it1 AND CREATE A SORT KEY IN &*k  *
.* THE SEE OR SEEID ATTRIBUTE MAY PROVIDE A CROSS-REFERENCE TERM IN &*r
.se @tg = i
.gs exatt print as dsm@iprt id as dsm@ids
.gs exatt see as dsm@see seeid as dsm@seei
.if &E'&*r eq 1 .se *x = ref
.*                 CREATE AN INDEX ENTRY.                              *
.if &L'&@it1 eq 0 .dsm#msg 14 '1(H)'
.'el .'pi &*x &*k &X'01.&#it1.&X'01.&*r.&X'010101
```

DSMIHD2

```
.*  DSMIHD2:  Tag =IH2  Attr = ID, PRINT, SEE, SEEID  Creates an        *
.*  index header (an index entry w/ no page numbers).                   *
.*  Gets the residual text into &@it2 and copies it also to &#it2.     *
.*********************************************************************
.gs scan @it2
.'se #it2 '&@it2
.*                THE SYMBOL &@it3 IS NULL FOR LEVEL 2 TERMS           *
.gs args 2       ''
.gs vars @ilevel @it3
.*          THE PRINT ATTRIBUTE MAY RESET &#it2 and SEE or SEEID MAY  *
.*               PROVIDE A CROSS REFERENCE TERM IN &*r                 *
.se @tg = i
.gs exatt print as dsm@iprt id as dsm@ids
.gs exatt see as dsm@see seeid as dsm@seei
.if &E'&*r eq 1 .se *x = ref
.*             CREATE AN INDEX ENTRY                                   *
.if &L'&@it1 eq 0 .or &L'&@it2 eq 0 .dsm#msg 14 '2(H)'
.'el .'pi &*x &*k &X'01.&@it1.&X'01.&#it2.&X'01.&*r.&X'0101
```

DSMIHD3

```
.*  DSMIHD3:   Attr = ID, PRINT   Creates an index header ( an index  *
.*  entry with no page numbers).  Gets the residual text into &@it3   *
.*  and copies it into &#it3 also.                                    *
.*********************************************************************
.gs scan @it3
.'se #it3 '&@it3
.se @ilevel = 3
.*                 THE PRINT ATTRIBUTE MAY RESET  &#it2               *
.se @tg = i
.gs exatt print as dsm@iprt id as dsm@ids
.*                 CREATE AN INDEX ENTRY                              *
.if &L'&@it1 eq 0 .or &L'&@it2 eq 0 .or &L'&@it3 eq 0
.th .dsm#msg 14 '3(H)'
.'el .'pi &*k &X'01.&@it1.&X'01.&@it2.&X'01.&#it3.&X'0101
```

DSMIM

```
.*  DSMIM: Intercepts the .IM control word, issues a message and      *
.*  saves the information for the imbed trace.  Does not trace imbeds *
.*  of SCRIPT/VS utility files.                                       *
.*********************************************************************
.se @ = index '-DSMUTTOC-DSMUTWTF-DSMUTDIM-' '-&*1.-'
.'if &@ ne 0 .'im &*
.th .me
.*                 &@nest@i IS THE IMBED FILE NESTING LEVEL           *
.se @nest@i = &@nest@i + 1
.*              ISSUE MSG W/ CURRENT PASS & IMBED FILE NESTING LEVEL  *
.se *a = 3 * &@nest@i
.se *a = substr '-----------------------------------------------' 1 &*a
.se *p = &
.se *page = substr '    &*p.' &L'&*p 5
.if &$TWO eq 1 .se *pass '(&LL@Pass &$PASS.)  '
.ty &*pass.&LL@Page.&*page.:  &LL@Imbdg &*a.> &U'&*
.*                    SAVE INFORMATION FOR THE IMBED TRACE            *
.if &@lastpass ne yes .go imbed
.se *a = 2 * &@nest@i
.se *a = substr '                                              ' 1 &*a
.se @imtrace( ) '&LL@Page &*p.&$TAB.&*a.&U'&*
.*                     IMBED THE FILE                                 *
...imbed
.'im &*
.se @nest@i = &@nest@i - 1
```

DSMINDEX

```
.*  DSMINDEX: Tag = INDEX  No Attr.  Formats the index.  Resets any   *
.*  open lists, etc.  Advances to next/od page.                       *
.*********************************************************************
.dsm#rset Index
.dsm#dupl
.'se @shead '&LL@Index
.*                 MAKE SURE THE GML INDEX HEADER MACRO IS PRESENT    *
.dm ieh /.dsmieh &*/
.dm dsmieh lib
.*                 USE THE .IX CONTROL WORD TO GENERATE THE INDEX     *
.sa
.dsm#styl two
.fo left
.ly off
.'ix &LL@Index
.ly mac
.re
```

DSMINDX1

```
.*  DSMINDX1:  Tag = I1  Attr = ID, PAGEREF  Produces a level 1 index *
.*  entry.  &@it1 contains the level one index term. &@it2 and &@it3  *
.*  are reset. &*t4 contains the page #, and is initialized to &$PS   *
.*******************************************************************
.gs scan @it1
.gs args 1        ''    ''    &$PS
.gs vars @ilevel @it2 @it3 *t4
.*  PROCESS PAGEREF - MAY RESULT IN &*x, CONTAINING A .PI PARAMETER   *
.*      OR IN THE SYMBOL &*T4 BEING NULLED OR RESET                   *
.se @tg = i
.gs exatt pg as dsm@pgrf pageref as dsm@pgrf id as dsm@ids
.*               CREATE THE INDEX ENTRY                               *
.if &L'&@it1 eq 0 .dsm#msg 14 1
.'el .'pi &*x &X'01.&@it1.&X'010101.&*t4.&X'01
```

DSMINDX2

```
.*  DSMINDX2:  Tag = I2  Attr = ID, PAGEREF, REFID   Produces a       *
.*  level 2 index entry with the current page number unless PAGEREF   *
.*  specified &@it2 contains the level two index term. &@it3 is       *
.*  reset. &*t4 contains the page number, and initialized to &$PS.    *
.*******************************************************************
.'se #it1 '&@it1
.gs scan @it2
.gs args 2        ''    &$PS
.gs vars @ilevel @it3 *t4
.*  PROCESS THE ATTRIBUTES - PAGEREF MAY RESULT IN  &*x, CONTAINING   *
.*  A .PI PARAMETER, OR IN THE SYMBOL &*t4 BEING NULLED OR RESET      *
.*  REFID and ID ATTRIBUTES ARE ALSO PROCESSED (MAY RESET &#it1)      *
.se @tg = i
.gs exatt pg as dsm@pgrf pageref as dsm@pgrf
.gs exatt refid as dsm@ridi id as dsm@ids                             *
.*                CREATE THE INDEX ENTRY
.se @it1 '&#it1
.if &L'&#it1 eq 0 .or &L'&@it2 eq 0 .dsm#msg 14 2
.'el .'pi &*x &X'01.&#it1.&X'01.&@it2.&X'0101.&*t4.&X'01
```

DSMINDX3

```
.*  DSMINDX3:  Tag = I3  Attr = ID, PAGEREF, REFID   Produces a level *
.*  3 entry using current page number unless PAGEREF was specified.   *
.*  &@it3 contains the level three index term.                        *
.*******************************************************************
.'se #it1 '&@it1
.'se #it2 '&@it2
.gs scan @it3
.gs args 3       &$PS
.gs vars @ilevel *t4
.*        PROCESS PAGEREF - MAY RESULT IN  &*x CONTAINING             *
.*           A .PI PARAMETER, OR IN &*t4 BEING NULLED OR RESET        *
.gs exatt pg as dsm@pgrf pageref as dsm@pgrf
.*     PROCESS ID AND REFID ATTRIBUTES - MAY RESET &*it1 AND &*it2    *
.se @tg = i
.gs exatt refid as dsm@ridi id as dsm@ids
.*                      CREATE THE INDEX ENTRY                        *
.if &L'&#it1 eq 0 .or &L'&#it2 eq 0 .or &L'&@it3 eq 0
.th .dsm#msg 14 3
.'el .'pi &*x &X'01.&#it1.&X'01.&#it2.&X'01.&@it3.&X'01.&*t4.&X'01
```

DSMIREF

```
.*  DSMIREF: Tag = IREF   Attr = REFID, PAGEREF, SEE, SEEID   Creates *
.*  an index entry from index terms saved for the 'id'.  The page ref *
.*  is the current page, unless PAGEREF was specified.               *
.*********************************************************************
.se *t4 = &$PS
.gs exatt refid as dsm@rfid
.if &L'&*id gt 7 .dsm#msg 8 I &*id
.th .se *id = substr '&*id.' 1 7
.*   IF &I1@.... EXISTS, WE CAN GENERATE AN INDEX ENTRY              *
.IF &E'&I1@&*id eq 0 .go unknown
.* DETERMINE WHAT LEVEL ENTRY WILL BE, & RECOVER &*it1, &*it2, &*ii3*
.se @ilevel = &E'&I1@&*id + &E'&I2@&*id + &E'&I3@&*id
.go index&@ilevel
...index3 .'se *t3 '&I3@&*id
...index2 .'se *t2 '&I2@&*id
...index1 .'se *t1 '&I1@&*id
.*              PROCESS THE PAGEREF, SEE, and SEEID ATTRIBUTES       *
.gs exatt pg as dsm@pgrf pgref as dsm@pgrf pageref as dsm@pgrf
.if &@ilevel gt 2 .go skip
.gs exatt see as dsm@see seeid as dsm@seei
.if &E'&*r eq 0 .go skip
.se *x = ref
.se *a = &@ilevel + 1
.'if &*a le 3 .'se *t&*a '&*r
...skip
.*                GENERATE THE INDEX ENTRY AND BE DONE WITH IT       *
.'pi &*x &X'01.&*t1.&X'01.&*t2.&X'01.&*t3.&X'01.&*t4.&X'01
.*              SAVE THE CURRENT PAGE NUMBER
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.se IX@&*id.() = &
.me
.*                 SAVE THE CROSS REFERENCE INFORMATION              *
...unknown
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.if &E'&IF@&*id eq 0 .se @xref@i() '.dsm#xrfi &*id
.th .se IL@&*id = &@xref@i(0)
.th .se IF@&*id = ?
.se IX@&*id.() = &
```

DSMLIREF

```
.*  DSMLIREF: Tag = LIREF Attr = PAGE, REFID  Inserts list item cross *
.*  references into document. Includes item identifier & page refer-  *
.*  ence unless PAGE attribute was specified. &*yesno will be "no" if *
.*  PAGE attribute was given. the REFID  attribute will reset &*id.   *
.*********************************************************************
.gs exatt refid as dsm@rfid page as dsm#yesn
.if &L'&*id gt 7 .dsm#msg 8 D &*id
.th .se *id = substr '&*id.' 1 7
.*   IF &D1@.... EXISTS, GENERATE A STRING CONTAINING THE IDENTIFIER  *
.se *D = 'D'
.if &E'&D1@&*id eq 0 .an &E'&d1@&*id eq 0 .go unknown
.if &E'&D1@&*id eq 0 .se *D = d
.su off
.se *cw = &$CW
.dc cw off
.se *p = &
.*           PERHAPS INCLUDE THE PAGE REFERENCE                       *
.if &*yesno eq yes .or &*p ne &&*D.P@&*id .an &*yesno ne no
.th .se *r ' &LL@onpge &&*D.P@&*id..'
.*              GENERATE THE TEXT OF THE LIST ITEM REFERENCE          *
.su on
&&*D.1@&*id..&*r.&$CONT.
.dc cw &*cw
.*              IF CROSS REFERENCING SAVE THE PAGE NUMBER             *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.se DX@&*id.() = &
.me
.*        IF  &D1@.... DOES NOT EXIST, USE A CANNED STRING INSTEAD    *
...unknown
.sv off
-- &LL@LI '&*id.' --&$CONT
.sv on
.*          SAVE THE CROSS REFERENCE INFORMATION                      *
.if &$PASS ne 1 .or &SYSVARX ne yes .me
.if &E'&DF@&*id eq 0 .se @xref@d() '.dsm#xrfd &*id
.th .se DL@&*id = &@xref@d(0)
.th .se DF@&*id = ?
.se DX@&*id.() = &
```

DSMLISTM

```
.*  DSMLISTM: Tag = L  Attr: TYPE, COMPACT,TERMHI,HEADHI,TSIZE        *
.*  Sets up for all types of lists.  Stacks current list parameters.  *
.*  &@hi@l - highlight for def term    &@in   - current indention     *
.*  &@ltype- list type                 &@id@l - "id" for this list    *
.*  &@in@l - indent for this list type &@item# - list item counter    *
.*  &@sk@l - amount of skip before list item                          *
.*********************************************************************
.*                DEFINE ATTRIBUTE PROCESSORS FOR THE LIST TAGS       *
.dm dsmlistm(&$LNUM.) off &$CW..dm @termhi /.se @hi@l = &*1/
.dm dsmlistm(&$LNUM.) off &$CW..dm @tsize /.se @in@l '&*/
.dm dsmlistm(&$LNUM.) off &$CW..dm @headhi /.se @hi@hd '&*1/
.* CONDITIONAL SKIP IN CASE WE'RE GOING FROM UNCOMPRESSED TO COMPRESSED
.sk &@sk@l c
.*         &@nest@l INDICATES CURRENT NESTING LEVEL AND CONTAINS      *
.*                  SAVED FORMATTING PARAMETERS OF NESTED LISTS.      *
.se *g = '&@ltype &@item# &@in
.se *h = '&@sk@l. &V'&@hi@l. &V'&@hi@hd.
.se @nest@l() '&*g '&@in@l.' &*h.   &@break.
.*   &@in DEFINES THE BASE INDENTION FOR THIS LEVEL OF LIST NESTING   *
.if &@nest@l(0) gt 1 .in &@in
.th .in +&@in@l
.se @in = &DH'&$IN.dh
.* DSM#LTYP RETURNS THE LIST TYPE IN @ltype AND THE ITEM IDENTIFIER   *
.*                   FOR THIS LEVEL OF NESTING.                       *
.dsm#ltyp &*1
.*   CHECK FOR COMPACT; IF PRESENT, &@sk@l WILL BE ZEROED             *
.se @break = ''
.se *a = index '1COMPACT1BREAK' '&E'&*2.&U'&*2.'
.if &*a eq 1 .se *c = 0
.if &*a eq 9 .se @break = BREAK
.se *b = index '1COMPACT1BREAK' '&E'&*3.&U'&*3.'
.if &*b eq 1 .se *c = 0
.if &*b eq 9 .se @break = BREAK
.*                SET THE FORMATTING PARAMETERS FOR THIS LIST TYPE    *
.gs args 0     &@hi@&@ltype &@in@&@ltype  &@hi@h      &*c &@sk@&@ltype
.gs vars @item#  @hi@l       @in@l        @hi@hd      @sk@l
.*   PROCESS TSIZE AND TERMHI ATTRIBUTES TO RESET &@in@l AND &@hi@l   *
.gs exatt tsize as @tsize thi as @termhi termhi as @termhi
.gs exatt hhi as @headhi headhi as @headhi
.se @in@l = &DH'&@in@l.dh
.*   ESTABLISH A TAB STOP FOR THE FIRST LINE OF EACH LIST ITEM.       *
.in &@in
.in +&@in@l
.se @li@tab = &DH'&$IN.dh
.* ENABLE LIST PART TAGS (LI OR DT, DD, DTHD AND DDHD OR GT AND GD    *
.aa lp dsmlpart
.if &@ltype eq g
.th .aa gt dsmgterm
.th .aa gd dsmgdef
.th .me
.if &@ltype ne d .aa li dsmlitem
.el .aa dt dsmdterm
.el .aa dd dsmddef
.el .aa dthd dsmdthd
.el .aa ddhd dsmddhd
```

DSMLITEM

```
.*  DSMLITEM: Tag = LI  No Attr  Formats list items controlled by       *
.*  symbol by  DSMLISTM:  &@id@l - "id" for this list type              *
.*  &@in   - current indention            &@item# - list item counter   *
.*  &@in@l - indent for this list type  &@sk@l - skip before item       *
.***********************************************************************
.*      INCREMENT THE LIST ITEM COUNTER AND INSERT SOME WHITE SPACE     *
.se  @item# = &@item# + 1
.sk &@sk@l
.*    FORCE PROPER INDENTION FOR THIS LEVEL OF LIST, THEN UNDENT 1st   *
.*         LINE TO ALIGN WITH TEXT PRECEDING THE LIST                   *
.in &@in
.in +&@in@l after 1
.*                  PROCESS THE 'ID' ATTRIBUTE,  IF PRESENT             *
.se @tg = d
.gs exatt id as dsm@ids
.*      INSERT THE IDENTIFIER INTO THE GUTTER CREATED BY THE UNDENT     *
&@id@l.
.is to &@li@tab min 1
```

DSMLPART

```
.*  DSMLPART: Tag = LP No Attr. Format list parts just like             *
.*  list items with no identifier.                                      *
.***********************************************************************
.sk &@sk@l
.in &@in
```

DSMLQUOT

```
.*  DSMLQUOT  Tag = LQ  No Attr  Establishes formatting environment   *
.*  for long quoted phrases.  @nest@l indicates current level of list *
.*  nesting and contains the formatting parameters for previous list  *
.***********************************************************************
.sk &@sk@q c
.se *h = '&@sk@l. &V'&@hi@l. &V'&@hi@hd.'
.se @nest@l( ) '&@ltype &@item# &@in '&@in@l.' &*h  &@break
.*    &@in IS THE BASE INDENTION FOR THE CURRENT LEVEL OF LIST NESTING *
.if &@nest@l(0) gt 1 .in &@in
.th .in +&@in@l
.gs args 0      q      0     0      &DH'&$IN.dh &DH'&@in@q.dh
.gs vars @item# @ltype @hi@l @hi@hd @in         @in@l
.se @break = ''
.*        ESTABLISH THE PROPER INDENTION FOR THE LONG QUOTE             *
.in &@in
.in +&@in@q
.ir +&@in@q
.bf lqfont =
```

DSMNOTE

```
.*  DSMNOTE: Tag = NOTE   No Attr.   Starts a note, which is a          *
.*  paragraph with  'NOTE' in front of it.                              *
.***********************************************************************
.sk &@sk@p
.bf hi2
&LL@Note.:&$CONT
.pf
&$RB.&$CONT
```

DSMOLIST

```
.* DSMOLIST: Tag = OL  No Attr. Calls DSMLISTM macro for ordered list *
.***********************************************************************
.dsmlistm o &*
```

DSMPARA

```
.*  DSMPARA:  Tag = P  No Attr. Spaces between paragraphs and indents *
.*  first line.                                                        *
.*********************************************************************
.sk &@sk@p
.il +&@in@p
```

DSMPARA1

```
.* DSMPARA1:  Tag = P No Attr. For the 1st paragraph after H0 or H1   *
.*********************************************************************
.sk &@sk@p
.aa p dsmpara
```

DSMPARA2

```
.* DSMPARA2: Tag = P  No Attr.  For the 1st paragraph after H2-4      *
.*********************************************************************
.sk &@sk@p
.aa p dsmpara
```

DSMPARA5

```
.*  DSMPARA5: Tag = P No Attr.  For 1st paragraph after H5 or H6.     *
.*  A colon is appended to the heading if no text has intervened.     *
.*********************************************************************
.bf hd&@para5@fnt
.if &$PN./&$LC eq &@h5line .ct :
.pf
.el .dsmpara
.aa p dsmpara
```

DSMPCONT

```
.* DSMPCONT: Tag = PC No Attr. Formats text as paragraph continuation *
.*********************************************************************
.sk &@sk@p
```

DSMPREF

```
.*  DSMPREF: Tag = PREFACE  No Attr.  Establishes formatting envir-   *
.*  onment for the preface.  Resets any open lists. Advances to       *
.*  next/odd page, generates level 1 heading for preface.             *
.*********************************************************************
.dsm#rset Pref
.dsm#dupl
.*                   SETS &@shead FOR THE RUNNING FOOTING             *
.'se @shead '&LL@Pref
.'h1 &LL@Pref
.*             NO INDENTION FOR 1ST PARAGRAPH AFTER IT                *
.aa p dsmpara1
```

DSMPROF3

```
.**********************************************************************
.*  COPYRIGHT: 5748-XX9 (c) COPYRIGHT IBM CORPORATION 1983            *
.*             THIS PRODUCT CONTAINS RESTRICTED MATERIALS OF IBM.     *
.* This Profile is for use with SCRIPT/VS and GML Starter Set REL 3   *
.**********************************************************************
.if &$DCF lt 3 .mg /S/The &$FNAM profile requires the use of Rel. 3
.se DSM@MAC@ lib
.if /&DSM@MAC@ ne /DSMGML3
.th .mg /S/The &$FNAM profile requires DSMGML3 MACLIB/
.**********************************************************************
.*   The following symbols define the amount of white space and       *
.*   indention surrounding various kinds of text:                     *
.*   &@in@d &@sk@d - definition list terms                            *
.*   &@in@f &@sk@f - figures                                          *
.*   &@in@o &@sk@o - ordered list items                               *
.*   &@in@g &@sk@g - glossary                                         *
.*   &@in@p &@sk@p - paragraphs                                       *
.*   &@in@q &@sk@q - long quotes                                      *
.*   &@in@s &@sk@s - simple list items                                *
.*   &@in@u &@sk@u - unordered list items                             *
.*   &@in@x &@sk@x - examples                                         *
.*   &@in@z &@sk@z - list items                                       *
.*          &@sk@n - footnote                                         *
.**********************************************************************
.gs args 10    2     4     4     0     3     4     4     2     0
.gs vars @in@d @in@f @in@z @in@o @in@p @in@q @in@s @in@u @in@x @in@g
.gs args .75   1     .75   .75   .75   1     .75   .75   1     .75
.gs vars @sk@d @sk@f @sk@z @sk@o @sk@p @sk@q @sk@s @sk@u @sk@x @sk@g
.**********************************************************************
.* The following symbols define the default highlight levels:         *
.* @hi@d - definition list terms   @hi@g - glossary list terms        *
.* @hi@h - definition list headings                                   *
.**********************************************************************
.gs args .75    1     3     2     2
.gs vars @sk@ls @sk@n @hi@h @hi@d @hi@g
.**********************************************************************
.*   The symbols &@olistnest and &@ulistnest indicate the sequence of *
.*   item identifiers for the items of ordered and unordered lists,   *
.*   respectively. The identifiers themselves are defined below using *
.*   the .DV control word. The &@olistnest and &@ulistnest are treat- *
.*   ed as rings if lists are nested beyond the level defined in      *
.*   these symbols. The symbols &@figplace and &@figwidth give the    *
.*   default value for figure placement and width.                    *
.**********************************************************************
.gs args 123456     123        top        page
.gs vars @olistnest @ulistnest @figplace @figwidth
.**********************************************************************
.*   Macro substitution, library lookup and tag processing must be    *
.*   enabled for GML tag processing.  The GML tag delimiter will be a *
.*   colon (:) and the GML end-tag delimiter will be colon-e (:e).    *
.**********************************************************************
.ms on
.ly mac
.dc gml : : e
.gs tag on
.gs rules (att novat stop nomsg) (noatt)
.if &L'&$CONT eq 0 .dc cont &X'03
.**********************************************************************
.* Define the specific character bullets for unordered lists.         *
.* FORM: .DV  @id@l@xy  /symbol function to create dingbat            *
.*      or                                                            *
.* FORM: .DV  @id@l@xy  FONT fontname /symbol function                *
.* where:                                                             *
.* x     is the list type -  either 'o', or 'u' or 'z'                *
.* y     is the nesting level number for which identifiers are being *
.*       defined. (1 through 3) are normally used.                    *
.* symbol: the symbol function used to produce the character to be    *
.*         used as the 'dingbat'.  (e.g. '&x''9f' or '&x''af' )       *
.*         Note: Remember to double any single quotes in the value    *
.* fontname:The font that the dingbat is to be printed in.            *
.*       Note: See .DV control word description for details.           *
.***********************************************************************
.if SYSOUT eq PRINT
.th .dv @id@l@ul /&X'af
.th .dv @id@l@u2 /&X'bf
.th .dv @id@l@u3 /&X'bfbf
.th .dv @id@l@u4 /&X'9f
.*
.el .dv @id@l@ul /o
.el .dv @id@l@u2 /-
.el .dv @id@l@u3 /--
.el .dv @id@l@u4 /o
.*
.df @pi@ul type('pi font sans serif' 8) codepage aftc0363
.*
.if &$PDEV eq 38PP .df @pi@ul type('pi sans serif' 8) codepage tlgpi363
.if SYSOUT eq PAGE .se @ulistnest '12345
.th .dv @id@l@ul              /&X'9f
.th .dv @id@l@u2 font @pi@ul /&X'db
.th .dv @id@l@u3 font @pi@ul /&X'ed
.th .dv @id@l@u4 font @pi@ul /&X'4d
.th .dv @id@l@u5 font @pi@ul /&X'da
.*
.el .dv @id@l@u5             /*
.***********************************************************************
.* Define the style of the list item numbers for ordered lists.        *
.* The '&@item#.' symbol will resolve to correct item number.          *
.***********************************************************************
.su off
.dv @id@l@o1           /&@item#..
.dv @id@l@o2           /&a'&@item#..
.dv @id@l@o3           /&@item#.)
.dv @id@l@o4           /&a'&@item#.)
.dv @id@l@o5           /&r'&@item#..
.dv @id@l@o6           /&r'&@item#.)
.*
.dv @id@l@o7           /&R'&@item#..
.dv @id@l@o8           /&A'&@item#..
.dv @id@l@o9           /.dsm#supr &@item#.
.*
.dv @id@l@z0           /*
.dv @id@l@z1           /*
.su on
.***********************************************************************
.* Define some special variables: &amp., &semi., and &gml              *
.***********************************************************************
.dv amp text /&
.dv gml text /:
.'dv semi text /;
.***********************************************************************
.*   Define some rules for use in the Starter Set                      *
.***********************************************************************
.dr @figrule
.dr @fnldr w .2mm
.***********************************************************************
.* If Headings are not to be numbered, but you want level 1 body       *
.* headings to be numbered (for example, "Chapter 1, Chapter 2, ...") *
.* set @bodyhead1 to the string which should precede the number.       *
.***********************************************************************
.*.se @bodyhead1 = 'Chapter
.***********************************************************************
.* Set linespacing, hyphenation and justification controls             *
.***********************************************************************
.ls all .90 1.1
.fv justify
.hy on minpt 2 maxpt 3 minword 5 ladder 2 range .8 1.2 noalg
.***********************************************************************
.*  Parameterize some spacing values so they can be changed easily.    *
.*  (two column gutter amt, length of footnote leader, footnote in-    *
.*  dention and lead out amount for 4250 titles on the title page.)    *
.***********************************************************************
.se @gutter = 4
.se @fnldrlen = 16
```

```
.se @ttllo = '1.2
.*****************************************************************
.*   Perform various GML initialization tasks:                   *
.*   .dsm#setv - Process SYSVAR variables                        *
.*   .dsm#sets - Create symbols for all literal text strings     *
.*****************************************************************
.dsm#setv
.dsm#sets
.*****************************************************************
.*  Also set space around headings for line devices.             *
.*****************************************************************
.se @hspbf = 0
.se @h0sp = 5
.se @h1sp = 3
.se @h2sk = 3
.se @h2sp = 2
.se @h3sk = 3
.se @h3sp = 2
.se @h4sk = 3
.se @h4sp = 2
.*****************************************************************
.*  Define fonts and super script style for various output devices   *
.*****************************************************************
.df hi0 font &$CHAR(1)
.se @suprstyl = parens
.se *go 'f&$CHAR(0)
.if SYSOUT eq PAGE .se @suprstyl = shifts
.th .se *go =' '
.if &$PDEV eq 3800 .or &$PDEV eq 1403 .se @suprstyl = nums
.go &$PDEV.&*go
...1403f1
...2741f1
.df hi1 us
.df hi2 os rpt 3
.df hi3 os rpt 3 us
.df hd0 os rpt 3 us up
.df hd1 os rpt 3 us up
.df hd2 os rpt 3 us up
.df hd3 os rpt 3 us
.df hd4 os rpt 3
.df hd5 os rpt 3
.df hd6  us
.df hd0toc os rpt 3
.df hd1toc os rpt 3
.df hd2toc font &$CHAR(1)
.df hd3toc font &$CHAR(1)
.go endfont
...3270f1
...3800f1
.df hi1 us
.df hi2 up
.df hi3 us up
.df hd0 up us
.df hd1 up us
.df hd2 up us
.df hd3 up us
.df hd4 up
.df hd5 up
.df hd6 us
.df hd0toc up
.df hd1toc up
.df hd2toc font &$CHAR(1)
.df hd3toc font &$CHAR(1)
.go endfont
...3800f2
.df hi1 font &$CHAR(1) us
.df hi2 font &$CHAR(2)
.df hi3 font &$CHAR(2) us
.df hd0 font &$CHAR(2) us up
.df hd1 font &$CHAR(2) us up
.df hd2 font &$CHAR(2) us up
.df hd3 font &$CHAR(2) us
.df hd4 font &$CHAR(2)
```

```
.df hd5 font &$CHAR(2)
.df hd6 font &$CHAR(1) us
.df hd0toc font &$CHAR(2)
.df hd1toc font &$CHAR(2)
.df hd2toc font &$CHAR(1)
.df hd3toc font &$CHAR(1)
.go endfont
...3800f3
.df hi1 font &$CHAR(2)
.df hi2 font &$CHAR(3)
.df hi3 font &$CHAR(3) us
.df hd0 font &$CHAR(3) us up
.df hd1 font &$CHAR(3) us up
.df hd2 font &$CHAR(3) us up
.df hd3 font &$CHAR(3) us
.df hd4 font &$CHAR(3)
.df hd5 font &$CHAR(3)
.df hd6 font &$CHAR(2)
.df hd0toc font &$CHAR(3)
.df hd1toc font &$CHAR(3)
.df hd2toc font &$CHAR(1)
.df hd3toc font &$CHAR(1)
.go endfont
...3800f4
.df hi1 font &$CHAR(2)
.df hi2 font &$CHAR(3)
.df hi3 font &$CHAR(4)
.df hd0 font &$CHAR(4) up
.df hd1 font &$CHAR(4) up
.df hd2 font &$CHAR(4) up
.df hd3 font &$CHAR(4)
.df hd4 font &$CHAR(3)
.df hd5 font &$CHAR(3)
.df hd6 font &$CHAR(2)
.df hd0toc font &$CHAR(3)
.df hd1toc font &$CHAR(3)
.df hd2toc font &$CHAR(1)
.df hd3toc font &$CHAR(1)
.go endfont
.*
...38PP
.*   388PP DEFAULT FONTS ARE THE SAME AS 4250 DEFAULTS EXCEPT FOR THE
.*   EXAMPLE FONT.
.df xmpfont type (prestige 9) codepage t1d0base
.*df figfont
.go 4250same
...4250
.df xmpfont type ('prestige elite')
.*df figfont
...4250same
.***********************************************************************
.*  Reset space around headings for page devices.                      *
.***********************************************************************
.se @hspbf = 1.3i
.se @h0sp = p14
.se @h1sp = p14
.se @h2sk = p20
.se @h2sp = p11
.se @h3sk = p18
.se @h3sp = p11
.se @h4sk = p14
.se @h4sp = p11
.*
.df @rh type (bold italic 9) up
.df @rf type (bold 9)
.df ieh type(14 bold
.*   HIGHLIGHT FONTS
.df hi0 type(normal
.df hi1 type(italic)
.df hi2 type(bold)
.df hi3 type(bold italic)
.df althi1 us
.df althi2 up
```

```
.df althi3 uc
.*   FOOTNOTE FONTS
.df super type (6)
.df fnt type (9)
.*   TITLE PAGE FONTS
.df title type (24 bold) up
.df author type(12
.df address type(10)
.df date type(11 italic)
.df docnum type(10 italic)
.df titlesec type(10 italic bold)
.*   HEADING AND TABLE OF CONTENTS FONTS
.df hd0 type (20 bold italic
.df hd1 type (20 bold
.df hd2 type (18 bold italic
.df hd3 type (14 bold
.df hd4 type (12 bold italic
.df hd5 type (bold
.df hd6 type (bold italic
.df hd0toc type (10 bold) up
.df hd1toc type (10 bold)
.df hd2toc type (10)
.df hd3toc type (10)
.*   FIGURE FONTS
.df figcap type ( 9 bold)
.df figdesc type(9)
.df lqfont type(9)
...endfont
.dsm#set
.*****************************************************************
.*   Set &@oquote to contain the characters for open quote delimiters *
.*   for successive levels of nested quotes;  Set @cquote to contain  *
.*   the characters to be used as close quote delimiters.             *
.*   Select  appropriate quote marks for 4250 devices.                *
.*****************************************************************
.se @oquote '"'"'"'"'"'"'"'"'
.se @cquote '"'"'"'"'"'"'"'"'
.gs args '"'   ''''   '"'   ''''
.gs vars oqq   oq    cqq   cq
.if SYSOUT eq PAGE
.th .se @oquote '&X'bd.&X'bb.&X'bd.&X'bb.&X'bd.&X'bb.&X'bd.&x'bb.'
.th .se @cquote '&X'be.&X'bc.&X'be.&X'bc.&X'be.&X'bc.&X'be.&x'bc.'
.th .gs args &x'bd &X'bb &x'be &x'bc
.th .gs vars oqq   oq    cqq   cq
.*****************************************************************
.* Adjust head-level definitions. If duplexing, H0 & 1, outjustified  *
.*****************************************************************
.se @head1 off
.if &SYSVARH ne no  .se *n = num
.th .gs hctr &SYSVARH
.el .se *n = nonum
.dh 0 nus nohy nup font hd0 spaf &@h0sp pa left sect tfont hd0toc ts nto
.dh 1 nus nohy nup font hd1 &*n spaf &@h1sp pa left sect tfont hd1toc ts
.dh 0 spbf &@hspbf
.dh 1 spbf &@hspbf
.dh 2 nus nohy nup font hd2 &*n skbf &@h2sk spaf &@h2sp tfont hd2toc
.dh 3 nus nohy nup font hd3 &*n skbf &@h3sk spaf &@h3sp tfont hd3toc
.dh 4 nus nohy nup font hd4 &*n skbf &@h4sk spaf &@h4sp
.dh 5 nus nohy nup font hd5 nbr
.dh 6 nus nohy nup font hd6 nbr
.*****************************************************************
.* Set line length to 6.8 inches unless user changed it or we're      *
.* formatting for one column.                                         *
.*****************************************************************
.se *a = &$LL
.ll
.if &*a ne &$LL .ll &*a
.el .if &SYSVARS ne one .ll 6.8i
.dsm#styl
.*****************************************************************
.*   Define running heading and running footing                       *
.*****************************************************************
.rh on
```

```
.bf @rh hi2
.'ce &$RB.&@sec.&$RB
.sp 2
.pf
.rh off
.*
.if &SYSVARD eq yes .go duplex
.rf on
.bf @rf hi0
.sp 2
.'sx f &X'01.&@shead.&X'01.&X'01.&$PS.&X'01
.rf off
.go assoc
.*
...duplex
.rf odd
.bf @rf hi0
.sp 2
.fo right
 &@shead.&$RB.&$RB.&$RB.&$RB.&$PS
.*
.rf even
.bf @rf hi0
.fo left
.sp 2
&$PS.&$RB.&$RB.&$RB.&$RB.&@stitle
.rf off
.*********************************************************************
.*   GML tag definitions and initial APF mapping.                   *
.*********************************************************************
...assoc
.aa abstract dsmabstr (noatt)
.aa address  dsmaddr  (noatt) dsmeaddr
.aa aline    dsm#cntx (noatt)
.aa appendix dsmappd  (noatt)
.aa author   dsm#cntx (noatt)
.aa backm    dsmbackm (noatt)
.aa body     dsmbody  (noatt)
.aa cit      dsmcit   (noatt) dsmecit
.aa date     dsm#cntx (noatt)
.aa dd       dsm#cntx (noatt)
.aa ddhd     dsm#cntx (noatt)
.aa dt       dsm#cntx (noatt)
.aa dthd     dsm#cntx (noatt)
.aa dl       dsmdlist (vat)   dsmelist
.aa docnum   dsm#cntx (noatt)
.aa fig      dsmfig dsmefig
.aa figcap   dsm#cntx (noatt)
.aa figdesc  dsm#cntx (noatt)
.aa figlist  dsmflist (noatt)
.aa figref   dsmfgref
.aa fn       dsmftnt          dsmeftnt
.aa fnref    dsmfnref
.aa frontm   dsmfront (noatt)
.aa gdoc     dsmgdoc       dsmegdoc
.aa gd       dsm#cntx (noatt)
.aa gl       dsmglist (vat)   dsmelist
.aa gt       dsm#cntx (noatt)
.aa hdref    dsmhdref
.aa hp       bf       (noatt) dsmehp
.aa hp0      dsmhp0   (noatt) dsmehp
.aa hp1      dsmhp1   (noatt) dsmehp
.aa hp2      dsmhp2   (noatt) dsmehp
.aa hp3      dsmhp3   (noatt) dsmehp
.aa h0       dsmhead0
.aa h1       dsmhead1
.aa h2       dsmhead2
.aa h3       dsmhead3
.aa h4       dsmhead4
.aa h5       dsmhead5
.aa h6       dsmhead6
.aa index    dsmindex (noatt)
.aa l        dsmlistm  (vat)   dsmelist
```

```
.aa li       dsm#cntx
.aa liref    dsmliref
.aa lp       dsm#cntx (noatt)
.aa lq       dsmlquot (noatt) dsmelqu
.aa note     dsmnote  (noatt)
.aa ol       dsmolist (vat)   dsmelist
.aa p        dsmpara  (noatt)
.aa pc       dsmpcont (noatt)
.aa preface  dsmpref  (noatt)
.aa psc      dsmpsc           dsmepsc
.aa q        dsmquote (noatt) dsmequot
.aa sl       dsmslist (vat)   dsmelist
.aa title    dsm#cntx
.aa titlep   dsmttlep (noatt) dsmettlp
.aa toc      dsmtoc   (noatt)
.aa ul       dsmulist (vat)   dsmelist
.aa xmp      dsmxmp           dsmexmp
.*********************************************************************
.*   Index tag definitions (dependent upon INDEX option).            *
.*********************************************************************
.if &$INDX eq 1 .gs args 1   2   3   dsmindx  dsmihd   dsmiref
.el .gs args             ' ' ' ' ' ' dsmidmmy dsmidmmy null
.aa i1   &*4.&*1
.aa i2   &*4.&*2
.aa i3   &*4.&*3
.aa ih1  &*5.&*1
.aa ih2  &*5.&*2
.aa ih3  &*5.&*3
.aa iref &*6
.gs args
.*********************************************************************
.*   END of PROFILE. .EF will terminate profile processing.          *
.*********************************************************************
.ef
.*********************************************************************
.*   EPIFILE: The following is processed after the primary input     *
.*   file has been completely processed. If SYSVARW was specified     *
.*   the DSM#WRIT macro is called to write out the ids to the file.   *
.*   If a cross reference has been requested, the DSM#XLST macro      *
.*   is called to generate the "id" cross references and imbed trace. *
.*********************************************************************
.if &E'&SYSVARW ne 0 .an &@lastpass eq yes .dsm#writ
.if &SYSVARX eq yes .an &@lastpass eq yes .dsm#xlst
.*
```

DSMPSC

```
.*  DSMPSC: Tag = PSC  Attr = PROCESS   Conditionally processes parts *
.*  of a document, based on the logical or physical device, or on the *
.*  value of SYSVARP.  Starts a conditional section.                  *
.*********************************************************************
.cs 9 off
.*ASSUME SECTION WILL BE INCLUDED, THEN PROCESS THE PROCESS ATTRIBUTE *
.cs 9 include
.gs exatt process as dsm@proc proc as dsm@proc p as dsm@proc
.cs 9 on
```

DSMQUOTE

```
.* DSMQUOTE:  Tag = Quote No Attr.  Establishes formatting envir-     *
.* onment for short in-line quotes.Surround text with quote marks.    *
.* &@nest@q indicates the level of quoted phrase nesting, and is      *
.* used to select the character to be used as the quote delimter      *
.*********************************************************************
.su off
.se @nest@q = &@nest@q + 1
.se *q = substr &@oquote &@nest@q 1
.su on
&*q.&$CONT
```

DSMSLIST

```
.* DSMSLIST: Tag = SL  Attr = COMPACT  Calls DSMLISTM to process list *
.*********************************************************************
.dsmlistm s &*
```

DSMTITLE

```
.*  DSMTITLE: Tag = TITLE  Attr = STITLE   Saves text in &@title      *
.*  array.  Sets &@stitle to title or STITLE for RF                   *
.*********************************************************************
.gs scan *line
.'se @title( ) '&*line
.dc asep 40
.'if &E'&@stinit eq 0 .'se @stitle '&@title(*)
.*                 PROCESS THE STITLE ATTRIBUTE                        *
.gs exatt stitle as dsm@sttl
```

DSMTOC

```
.*  DSMTOC:  Tag = TOC  No Attr  Formats the table of contents.       *
.*  Reset any open lists, etc.  Advance to next/odd page.             *
.*********************************************************************
.dsm#rset ToC
.dsm#dupl
.*    SETS &@shead TO 'Table of Contents' FOR THE RUNNING FOOTING     *
.'se @shead '&LL@ToC
.*    THE .TC CONTROL WORD WILL GENERATE THE TABLE OF CONTENTS        *
.'tc &LL@ToC
```

DSMTTLEP

```
.*  DSMTTLEP: Tag = TITLEP No Attr. Establishes formatting environ-   *
.*  ment for collecting information for the title page.  The title    *
.*  page is generated by the :ETITLEP tag through the DSM#TIPG macro. *
.*********************************************************************
.se @state = TtlPg
.*                      ENABLE THE TITLE PAGE TAG GROUP               *
.aa author  dsmauthr
.aa date    dsmdate
.aa docnum  dsmdcnum
.aa title   dsmtitle
.*          INITIALIZE THE FOLLOWING SYMBOLS IN CASE THEY AREN'T SET  *
.*                                BY A SUBSEQUENT TITLE PAGE TAG.     *
.gs args 0        ''       ''        ''       ''        ''
.gs vars @addctr @author @address @docnum @docdate @title
```

DSMULIST

```
.* DSMULIST:  Tag = UL  No Attr. Calls DSMLISTM to process the list   *
.*********************************************************************
.dsmlistm u &*
```

DSMXMP

```
.*  DSMXMP: Tag = XMP  Attr = DEPTH  Establishes the formatting envir-*
.*  onment for an example.  These are in-line text, kept together     *
.*********************************************************************
.if &@state ne open .dsm#msg 4 &$TAG &@state
.th .me
.se @state = Exmpl
.*  BREAK TO FORCE UNPROMOTED TEXT INTO COLUMN AND SAVE ENVIRONMENT   *
.br
.sa
.*    SPELLING CHECKING, HYPHENATION AND FORMATTING ARE ALL OFF       *
.sv off
.hy off
.fo off
.if &$ENV eq KP .kp off
.bf xmpfont =
.in +&@in@x
.*  START A KEEP TO PREVENT THE XMP FROM BEGIN BROKEN ACROSS COLUMNS  *
.sk &@sk@x
.kp on
.ls all 1.0
.ws
.es
.*  PROCESS THE DEPTH ATTRIBUTE INSIDE THE KEEP                       *
.gs exatt depth as sp
```

Glossary

Glossary terms are defined as they are used in this book. Element names and tags, and attribute names and labels are, for the most part, not included in the glossary. Descriptions of them can be found in the body of this manual. If you cannot find the term you are looking for, refer to the index or to the *Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

all-points addressability: The capability to address, reference, and position text, overlays, and images at any defined point on the printable area of a sheet. See page device and contrast with line device.

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 substitution 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.

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 string of letters and numerals that stands for the name of an attribute. An attribute's label is entered in the source document when specifying the attribute's value.

balancing: In multicolumn formatting, the process of making column depths on a page approximately equal.

baseline: An imaginary horizontal line upon which most of the letters in a line of text appear to rest.

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 that contains the main text.

boldface: A heavy-faced type. Also, printing in this type.

break: An interruption in the formatting of input lines so that the next input line is printed on a new output line.

call: Used in reference to macros. It means to invoke the macro.

caps: Capital letters. See also *initial caps.*

caption: Title of an illustration.

case sensitive: Whether a group of letters is uppercase or lowercase has relevance. ABC is different from Abc which is different from ABc.

centimeter (cm): A measurement equal to 0.39 inch. 100 cm = 1 meter (m).

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 4.511 mm (0.1776 in.) used in measuring typographical material.

CMS: An interactive processor that operators within VM/370.

column balancing: The process of redistributing lines of text among a set of columns so that the amount of text in each column is as equal as possible.

column line length: The width of each text column on a page. Specified with the .CL [Column Line Length] control word. (In multicolumn formatting, all columns on the page usually have the same line length.)

command: A request from a terminal or specification 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.

comment: A control word line that is ignored by SCRIPT/VS. Such lines begin with either .* or the .CM [Comment] control word.

composition: The act or result of formatting a document.

concatenation: The forming of an output line that contains as many words as the column line length 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.

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.

default: 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 0.3759 mm (0.0148 inch). There are 12 Didot points to a cicero. (See also cicero and point.)

document: (1) A data medium and the data recorded on it, that generally has permanence and that can be read by man or machine. (2) A unified collection of information pertaining to a specific subject or related subjects. (3) In word processing, a collection of one or more lines of text that can be named and stored as a separate entity. See also *output document* and *source document.*

document library: A set of VSM data sets, accessible in a batch environment, which contain documents and related files.

document administrator: One who is responsible for defining markup conventions and procedures for an organization.

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.

eject: In formatting, a skip to the next column or page.

element: Any part of a document: a single character or a word or a sentence. Also refers to any part of a document you can identify with a GML tag (tagged element), such as a paragraph or figure or heading.

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.

enable: Used in reference to a tag. Means that the tag is mapped to its appropriate APF.

fill character: The character that is used to fill up a space; for example, blanks used to fill up the space left by tabbing.

folio: Page number.

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) To be formatted in a location different from its location in the source file.

flush: Having no indention.

fold: (1) To translate the lowercase characters of a character string into uppercase. (2) To place that portion of a line that does not fit within a column on the next output line.

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).

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.

formatting mode: In document formatting, the state in which input lines are concatenated and the resulting output lines are justified.

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 can be used as a catch-all category for documents that 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.

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 (:).

gutter: In multicolumn formatting, the space between columns.

hanging indention: The indention of all lines of a block of text following the first line (which is not indented the same number of space). Specified with the .OF [Offset] or .UN [Undent] control word.

head-level: The typeface and character size associated with the words standing at the beginning of a chapter or chapter topic.

heading: Words located at the beginning of a chapter or section or at the top of a page. See also *head-level* and *running heading*.

hexadecimal: Pertaining to a number system based on 16, using the sixteen digits 0 - 9, A - F. For example, hexadecimal 1B equals decimal 27.

highlighting: Emphasis associated with a document element. In formatting, highlighting is usually expressed by changing font, overstriking, underscoring, and/or capitalizing the highlighted element.

horizontal justification: The process of redistributing the extra horizontal white space at the end of the line of text in between the words and letters of the line so as to exactly fill the width of the column with the text.

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. Specified with the .IN [Indent], .IR [Indent Right], .IL [Indent Line], .OF [Offset] and .UN [Undent] control words. See also *hanging indention.*

initial caps: Capital letters occurring 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.

initialize: This is a general programming term which means to set everything up correctly at the the beginning before you actually do any processing/ For the starter set it means doing things such as mapping tags to APFs and setting up symbol names and values.

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 upward to the right.

JCL: Job control language.

job control statement: A statement that provides an operating system with information about the job being run.

justification: The process of inserting 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.

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.

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.)

leader: (1) Dots or hyphens (as in a table of content) used to lead the eye horizontally. (2) The divider between text and footnotes on a page (usually a short horizontal rule).

left-hand page: The page on the left when a book is opened; usually even-numbered.

line device: Any of a class of printer that accept one line of text from the host system at a time. SCRIPT/VS supports such line devices as the 1403, 2741 and 3800.

line space: The vertical distance between the baseline of the current line and the baseline of the previous line.

lowercase: Pertaining to small letters as distinguished from capitals; for example, a, b, g rather than A, B, G.

maclib: See *macro library*

macro: See *macro instruction.*

macro library: A collection of macros. The form the library takes will vary by environment, being a MACLIB in CMS, a PDS in TSO and so on.

macro instruction: (1) An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language. (2) 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.

map: Associate a tap with an APF using the .AA [Associate APF] control word.

mark up: (verb) (1) To determine what information should be added to a document that would enable a person or system to process it. (2) To insert processing information into a source document.

markup: (noun) Information added to a document that enables a person or system to process it. Markup can describe the document's characteristics, or it can specify the actual processing to be performed. In SCRIPT/VS, markup consists of GML tags, attribute labels and values, and control words.

markup/content separator: A delimiter used in GML markup which indicates the end of the markup and the beginning of the text. The default markup content separator for SCRIPT/VS is a period (.).

meter (m): Basic unit of linear measurement.

millimeter (mm): One-thousandth of a meter. There are 10 millimeters in one centimeter. (25.4 millimeters = 1 inch.)

offset: (verb) To indent all lines of a block of text, except the first line. (noun) The indention of all lines of a block of text following the first line.

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 machinereadable 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.

page printer: Any of a class of printer that accept composed pages, constructed of composed text and images, among other things. SCRIPT/VS supports the 4250 printer, IBM 3820 Page Printer, and the 3800 Printing Subsystem Model 3, which are all page printers.

page segment: A datastream object containing composed text and images, prepared before formatting and included in a document when it is printed.

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.

pel: (Picture element) The unit of horizontal measurement for the IBM 3800 Printing Subsystem, the IBM 3820 Page Printer, and the 4250 printer. One pel equals approximately 1/180th of an inch on the 3800 Model 1, 1/240th of an inch on the 3800 Printing Subsystem Model 3 and 3820 Page Printer and 1/600th of an inch on 4250 printer.

pica: A unit of about 4.224 mm (0.1663 in. used in measuring typographical material. Similar to a Cicero in the Didot point system.

pitch: A number that represents the amount of horizontal space a font's character occupies on a line. For example, 10-pitch means 10 characters per inch, or each is 0.1 (1/10) inches wide. 12-pitch means 12 characters per inch.

point: (1) A unit of about 0.3759 mm (1/72 in.) used in measuring typographical material. There are twelve points to the pica. (2) In the Didot point system, a unit of 0.3759 mm. There are twelve Didot points to the Cicero.

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 source documents. When processing GML markup, the profile usually contains the association of GML with APFs, and the symbol settings that define the formatting style.

proportional spacing: The spacing of characters in a printed line so that each character is allotted a space proportional to the character's width.

ragged right: The unjustified right edge of text lines. See also *justification.*

required blank: A character that prints as a blank, but does not act as a word separator.

residual text: The line of text following the markup/content separator of a GML tag

right-hand page: The page on the right when a book is opened; usually odd-numbered.

rule: A solid black rectangle of a given width, extending horizontally across the column or vertically down the column.

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).

If the SEC attribute is specified on the :GDOC tag, the starter set formats the security line as a running heading.

SCRIPT/VS: The formatter component of the Document Composition Facility. SCRIPT/VS provides capabilities for text formatting and document management, macro processing and symbol substitution, and GML tag recognition and processing.

set: This term is used in reference to a symbol. It implies the .SE [Set Symbol] control word.

source document: A machine-readable collection of lines of text or images that is used for input to a computer program.

In this manual, the terms source document, source file, and source data set all mean the same thing.

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 that SCRIPT/VS can interpret as a value (numeric, character string, or control word) or as another symbol.

SYSVAR: An option of the SCRIPT command that 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 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. "What type of document is this?." "Type" is sometimes referred to as "document type," "element type," or "GML type."

typeface: (1) A specific type style, such as Univers or Press Roman. (2) One of the many attributes of a font, others for example, being size and weight.

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.)

uppercase: 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 space: The horizontal white space placed between words. This is sometimes referred to as an *interword blank*.

Index

Document Composition Facility:
GML Starter Set Implementation Guide
Order No. SH35-0050-2

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.

You may use this form to communicate your comments about this publication, its organization, or subject matter with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

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 the IBM branch office serving your locality.*

	Yes	No
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐
• How do you use this publication:		
As an introduction to the subject?	☐	
For advanced knowledge of the subject?	☐	
To learn about operating procedures?	☐	
As an instructor in class?	☐	
As a student in class?	☐	
As a reference manual?	☐	

• What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

SH35-0050-2

Reader's Comment Form

Fold and tape   **Please Do Not Staple**   Fold and tape

Attention: Information Development
Department 580

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40   ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Products Division
P. O. Box 1900
Boulder, Colorado 80301

Fold and tape   **Please Do Not Staple**   Fold and tape

IBM

Document Composition Facility GML Starter Set Implementation Guide Printed in U.S.A. SH35-0050-2

Document Composition Facility:
GML Starter Set Implementation Guide
Order No. SH35-0050-2

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.

You may use this form to communicate your comments about this publication, its organization, or subject matter with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

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 the IBM branch office serving your locality.*

	Yes	No
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐
• How do you use this publication:		
As an introduction to the subject?	☐	
For advanced knowledge of the subject?	☐	
To learn about operating procedures?	☐	
As an instructor in class?	☐	
As a student in class?	☐	
As a reference manual?	☐	

• What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

SH35-0050-2

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

Attention: Information Development
Department 580

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Products Division
P. O. Box 1900
Boulder, Colorado 80301

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM

Document Composition Facility GML Starter Set Implementation Guide Printed in U.S.A. SH35-0050-2

Document Composition Facility:
GML Starter Set Implementation Guide
Order No. SH35-0050-2

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.

You may use this form to communicate your comments about this publication, its organization, or subject matter with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

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 the IBM branch office serving your locality.*

	Yes	**No**
• Does the publication meet your needs?	☐	☐
• Did you find the information:		
Accurate?	☐	☐
Easy to read and understand?	☐	☐
Easy to retrieve?	☐	☐
Organized for convenient use?	☐	☐
Legible?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐
• How do you use this publication:		
As an introduction to the subject?	☐	
For advanced knowledge of the subject?	☐	
To learn about operating procedures?	☐	
As an instructor in class?	☐	
As a student in class?	☐	
As a reference manual?	☐	

• What is your occupation?

Comments:

If you would like a reply, please give your name and address.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail them directly to the address on the back of the title page.)

SH35-0050-2

Reader's Comment Form

Fold and tape | **Please Do Not Staple** | Fold and tape

Attention: Information Development
Department 580

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Information Products Division
P. O. Box 1900
Boulder, Colorado 80301

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM

Document Composition Facility GML Starter Set Implementation Guide Printed in U.S.A. SH35-0050-2

Document Composition Facility
Generalized Markup Language
Implementation Guide

SH35-0050-2

File Number S370-20
Printed in U.S.A.

SH35-0050-2