IBM

# Document Composition Facility: Generalized Markup Language Starter Set User's Guide

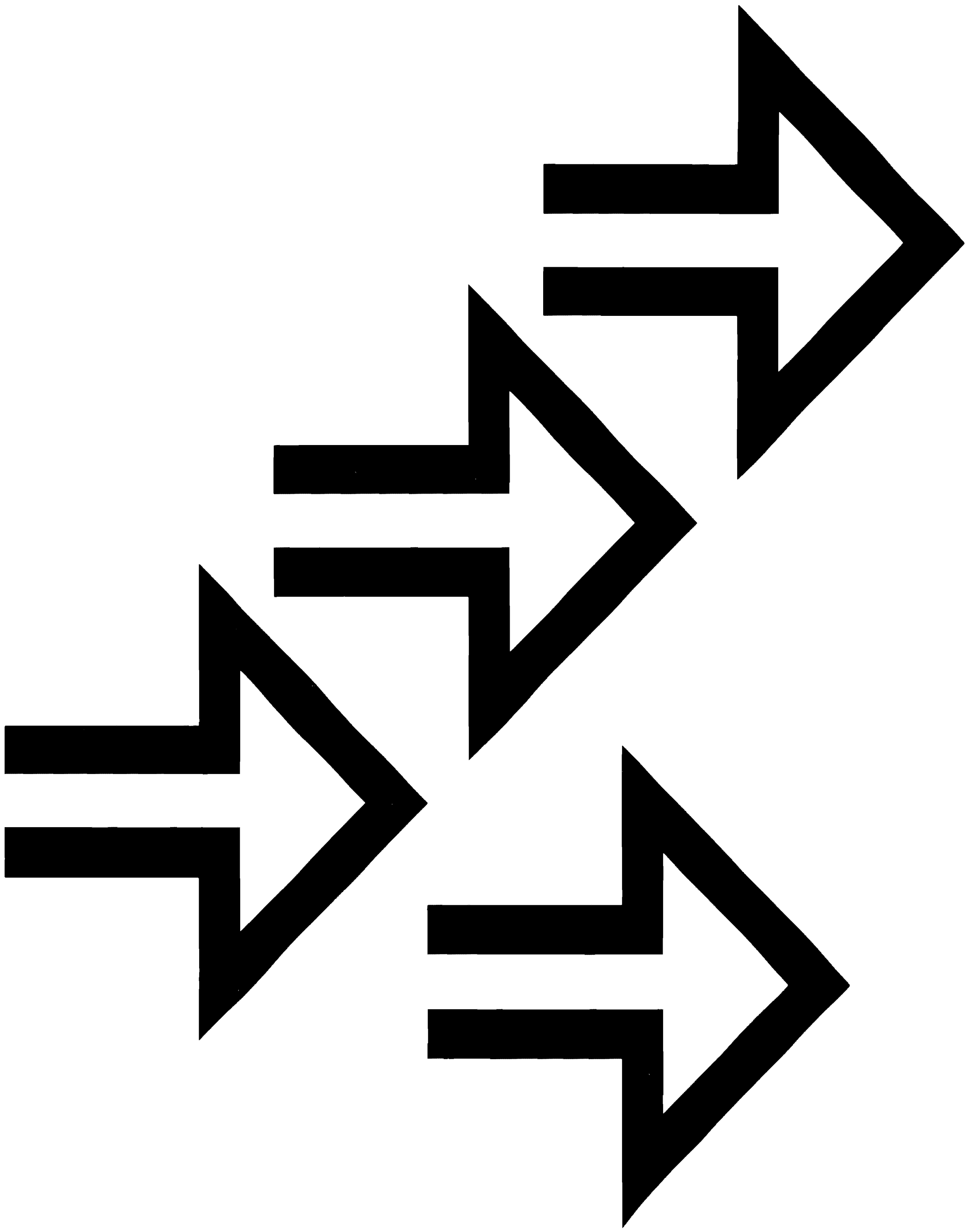
Licensed Program

Program Number 5748-XX9

Release 3.1

# Document Composition Facility: Generalized Markup Language Starter Set User's Guide

Licensed Program



Program Number 5748-XX9

Release 3.1

This publication was produced using the IBM Document Composition Facility (program number 5748-XX9). Masters were printed on the IBM 3820 Page Printer.

**Fifth Edition (May 1987)**

This edition contains information from and makes obsolete the *Document Composition Facility: Generalized Markup Language Starter Set User's Guide*, SH20-9186-03. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to Release 3.1 of the IBM Document Composition Facility Licensed Program, 5748-XX9, and to any subsequent releases of that program until otherwise indicated in new editions or technical newsletters.

# Preface

The IBM Document Composition Facility is a text processing program; its main component is the text formatter, called SCRIPT/VS.

One of the major features of SCRIPT/VS is the ability to develop and use the Generalized Markup Language (GML). IBM supplies, with the Document Composition Facility, a *starter set* of GML, to give users a starting point in using GML and developing their own GML to meet their specific needs.

This book is intended for new users of the Document Composition Facility and particularly for new users of GML. This book teaches you how to use GML in general and how to use the starter set in particular, and prepares you to use any specific GML that your organization might develop.

**Note: Central Programming Service support and maintenance are available for the Generalized Markup Language (GML) starter set. Support and maintenance, however, *are not* available if the starter set has been modified in any way. If you do make modifications, it is recommended that you also maintain an *unmodified* starter set for diagnostic purposes.**

References to the IBM 3800 Printing Subsystem in this book refer to the IBM 3800 Model 1 and the IBM 3800 Model 3 and Model 6 in all points addressability mode unless otherwise noted.

This book has the following major sections:

- "Introduction" is a brief introduction to the Generalized Markup Language starter set and describes what you need to know before you can get started using GML.
- "Part One: Tag Guide" describes the use of the starter set GML tags to mark up a document.
- "Part Two: Formatting Your Document" describes what you need to know in addition to the tags themselves to take full advantage of the starter set and SCRIPT/VS.
- "Part Three: Tag Reference" contains a reference summary for the starter set tags.
- "Appendixes" contains information on how to create and make changes to a file in the CMS, TSO, and ATMS environments; a sample GML document showing the marked-up source file and the formatted output for that file; the solutions to the exercises; and for users who are already familiar with earlier releases of the starter set, information on the new features of the Release 3 starter set.

## *Related Publications*

When you have become familiar with using GML from this book, you will want to use the books listed below for further information.

**For information on processing of starter set tags, see:**

*Document Composition Facility: Generalized Markup Language Starter Set Reference*, SH20-9187.

**For quick GML lookup, see:**

*Document Composition Facility: Generalized Markup Language Starter Set Quick Reference*, SX26-3719.

ATMS-III users will also need:

*ATMS-III: Terminal Operator's Guide*, SH20-2425

**For a description of SCRIPT/VS control words and command options, see:**

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

*Document Composition Facility: SCRIPT/VS Language Reference*, SH35-0070

**For a description of designing GML for specific requirements, refer to:**

*Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide*, SH35-0050

*Document Composition Facility: Generalized Markup Language Concepts and Design Guide*, SH20-9188

**For a description of the PL/I post processors supplied with DCF refer to:**

*Document Composition Facility Post-Processor Examples*, S544-3484.

This booklet describes the sample PL/I post processors supplied with DCF that create overlays from SCRIPT/VS output, create multiple-up pages from SCRIPT/VS output and insert variable text into SCRIPT/VS output. It describes the processing done by the sample programs and gives instructions for running the post processors in the VM, MVS, and VSE environments. The post processor examples provided in this publication apply only to page printers.

**For a general description of the Document Composition Facility and the Document Library Facility, see:**

*Document Composition Facility and Document Library Facility General Information*, GH20-9158.

**Student courses include:**

- *Using the Document Composition Facility*, SR21-0508 (Training Course 32291).

  This computer-based training course teaches the student how to mark up and format documents using DCF. The course consists of six sessions which will train the student in the use of GML and the SCRIPT command. It also provides an introduction to the SCRIPT/VS control words.

- *Using DCF with the 4250 Printer*, SR20-8486 (Training Course 32908).

  This self-study course introduces some functions of DCF and how they are used with the IBM 4250 Printer. This course is intended for text programmers and administrators interested in using the IBM 4250 Printer with DCF.

- *Using DCF with Page Printers*, SR21-1211 (Training Course 32243).

  This self-study course introduces some functions of DCF and how they are used with page printers. This course is intended for text programmers and administrators interested in using page printers with DCF.

- *Document Composition Facility—Release 3 (SCRIPT/VS) for Document Administrators and Text Programmers*, SR20-7525.

  This Independent Study Program explains techniques used in the GML starter set. It also provides information and techniques for extending or replacing the starter set.

The primary purpose of this course is to teach a person how to create GML tags and write APFs and how to modify the GML starter set to specific text processing needs. In addition, the course gives examples of how SCRIPT/VS control words, symbols, and logic can be applied to meet other text processing needs.

- *Document Composition Facility (SCRIPT/VS) Student Text*, SC20-1894.

  This self-study course teaches how to use DCF. This course has been divided into six units called modules:

  |  |  |
  |---|---|
  | Module 1: | Introduction |
  | Module 2: | Markup |
  | Module 3: | Organizing a Document |
  | Module 4: | Arranging Text |
  | Module 5: | Using Control Words |
  | Module 6: | Using the SCRIPT Command |

## *Publication Library Guide for the Document Composition Facility*

The following table is a library guide to the publications for the Document Composition Facility (DCF). The publications are listed as they relate to user tasks.

| User Tasks | Typical Audience | Recommended Books | Brief Description |
|---|---|---|---|
| Planning and introducing DCF/DLF | Users, system planners | *DCF and DLF General Information* (GH20-9158) *DCF (SCRIPT/VS) Student Text* (SC20-1894—Training Course) | Provides a general overview of text processing, library facility, and available books |
| Formatting documents (using the GML starter set) | Novice user to experienced end users | *DCF: GML Starter Set User's Guide* (SH20-9186) *DCF: GML Starter Set Reference* (SH20-9187) *DCF Messages* (SH35-0048) *Using the Document Composition Facility* (SR21-0508—Training Course 32291) | Provide an introduction to the GML starter set, describe the GML starter set tags and SCRIPT/VS messages |
| Formatting documents (using SCRIPT/VS control words) | Knowledgeable to experienced end users | *DCF: SCRIPT/VS Text Programmer's Guide* (SH35-0069) *DCF: SCRIPT/VS Language Reference* (SH35-0070) *DCF Messages* (SH35-0048) *Using DCF with the 4250 Printer* (SR20-8486—Training Course 32908) *Using DCF with Page Printers* (SR21-1211—Training Course 32243) *DCF—Release 3 (SCRIPT/VS for Document Administrators and Text Programmers* (SR20-7525—Training Course) | Describe the function and use of all SCRIPT/VS control words, SCRIPT/VS macros, SCRIPT/VS diagnostic aids, and the formatting features and messages of SCRIPT/VS |
| Modifying GML starter set[1] | Document administrator and text programmer[2] | *DCF: GML Starter Set Implementation Guide* (SH35-0050) *DCF: GML Starter Set User's Guide* (SH20-9186) *DCF: GML Starter Set Reference* (SH20-9187) *DCF: SCRIPT/VS Text Programmer's Guide* (SH35-0069) *DCF: SCRIPT/VS Language Reference* (SH35-0070) *DCF—Release 3 (SCRIPT/VS for Document Administrators and Text Programmers* (SR20-7525—Training Course) | Contain material on: GML starter set tags, SCRIPT/VS control words, and how to modify the GML starter set |

| User Tasks | Typical Audience | Recommended Books | Brief Description |
|---|---|---|---|
| Creating GML application processing functions | Document administrator and text programmer[2] | *DCF: GML Starter Set Implementation Guide* (SH35-0050) *DCF: GML Starter Set User's Guide* (SH20-9186) *DCF: GML Starter Set Reference* (SH20-9187) *DCF: SCRIPT/VS Text Programmer's Guide* (SH35-0069) *DCF: SCRIPT/VS Language Reference* (SH35-0070) *DCF: GML Concepts and Design Guide* (SH20-9188) *Using the Document Composition Facility* (SR21-0508—Training Course 32291) *DCF—Release 3 (SCRIPT/VS for Document Administrators and Text Programmers* (SR20-7525—Training Course) | Provide information on: how to design your own GML, GML concepts, GML starter set tags, SCRIPT/VS control words, and usage guidelines |
| Installing, modifying, and maintaining DCF | Systems programmer | DCF Program Directory *DCF: SCRIPT/VS Text Programmer's Guide* (SH35-0069) *DCF: SCRIPT/VS Language Reference* (SH35-0070) *DCF Diagnosis Guide* (SY35-0067) *DCF Diagnosis Reference* (LY35-0068) *DCF Messages* (SH35-0048) *DCF—Release.3 (SCRIPT/VS for Document Administrators and Text Programmers* (SR20-7525—Training Course) | Give information on error isolation, program tailoring, and use of SCRIPT/VS |

**Note:** As a convenience to Document Composition Facility users, the following aids are available:

*Document Composition Facility: Text Programmer's Quick Reference*, SX26-3723
*Document Composition Facility: GML Starter Set Quick Reference Summary*, SX26-3719.
*Document Composition Facility SCRIPT/VS*, SH35-0086 (three-ring binder).

[1] **Central Programming Service support and maintenance are provided *only* on the unmodified GML starter set.**

[2] The document administrator is responsible for defining markup conventions and procedures for an organization. The text programmer implements application processing functions (APFs) that provide the processing specified by the document administrator.

# Table of Contents

# List of Illustrations

# List of Tables

**Central Programming Service support and maintenance are available for the Generalized Markup Language (GML) starter set. Support and maintenance, however, *are not* available if the starter set has been modified in any way. If you do make modifications, it is recommended that you also maintain an *unmodified* starter set for diagnostic purposes.**

# Introduction

This book assumes you already know how to communicate with an operating system (computer), and how to create and edit (make changes to) files. If you do need information on creating files, see "Appendix A. Creating and Editing a File" on page 149 for a brief explanation of how to create and edit a file in the CMS, TSO, and ATMS-III environments.

For complete instructions on how to communicate with a computer and create and edit files in these environments, see the following books:

For the CMS environment, see:

- *IBM Virtual Machine/System Product: CMS User's Guide*, SC19-6210
- *IBM Virtual Machine/System Product: System Product Editor User's Guide*, SC24-5220

For the TSO environment, see:

- *OS/VS2 TSO Terminal User's Guide*, GC28-0645
- *Interactive System Productivity Facility/Program Development Facility for MVS: Program Reference*, SC34-2089

For the ATMS-III environment, see:

- *Advanced Text Management System-III Terminal Operator's Guide*, SH20-2425.

## *Phrases Used in this Book*

A number of descriptive phrases are used throughout this book. We'll introduce some of the most commonly used ones here to help you understand them when you come upon them later. You can refer to them as needed. There's also a glossary at the back of this book with an extensive listing of words and phrases used.

| | |
|---|---|
| **Document element** | Any part of a document: a single character, a word, or a sentence. Also refers to any part of a document you can identify with a GML tag, such as a paragraph, figure, or heading. |
| **GML tag** | In GML markup, an abbreviated name used to identify a document element. In the starter set of GML, for example, the :P tag identifies the beginning of a paragraph. |
| **Paragraph unit** | Paragraphs and other document elements that have the same structure as a paragraph, such as notes and paragraph continuations. |
| **Implied paragraph** | Paragraphs for which no P (paragraph) tag has been entered, such as definition descriptions, list parts, figure descriptions, and footnotes. |

| | |
|---|---|
| **Attributes** | Additional information used with tags. Attributes are placed immediately following the tag. There are two kinds of attributes we use with GML: • Value attributes—these are one-word descriptions. • Labeled attributes—these attributes always take an equal sign (=). The attribute is given, followed by =, followed by the value, like this: PLACE=top where PLACE is the attribute and top is its value. |
| **Default** | The value used if you do not specify a value. |
| **GML Delimiter** | The very first character of a tag. The colon (:) is the starter set delimiter. It calls attention to the fact that "This is a tag!" Some tags need to show where they end. End tags consist of the tag name prefixed with the end tag delimiter. The starter set end tag delimiter is :e. |
| **Markup Content Separator** | This signals the end of the tag and any attributes specified with the tag. A period (.) is the starter set markup content separator. |

Now, before we show you how to use GML, there are a few general things you need to know; they're covered in the following pages.

# Chapter 1. What is GML?

To use any text processing program, you must create a document that includes information in addition to the text itself, that tells the text processor what to do. This information is called *markup*.[3]

There are two basic ways of doing markup using the Document Composition Facility (DCF):

- One way is to tell the text processing program (which in our case is SCRIPT/VS) exactly how you want something, such as a numbered list, to look on the page. You do this with *control words*. When you use control words, you have to tell the text processing program exactly how many lines to skip, how many characters to indent, how far to offset text, and so on.

- The other way is to describe what something is (rather than what it looks like on the page) and let the text processing program take care of what it looks like on the page. This approach is called *Generalized Markup Language* or *GML*.

  GML is a *general* SCRIPT/VS capability; the *starter set* of GML tags provided with the Document Composition Facility is a specific set of GML markup designed for *general documents*.

  Using GML, you describe what something is by including *tags* in the text.

  Then, when you request formatting for your document, you also give SCRIPT/VS the name of a *profile* to be used when formatting. The profile is a file that is processed before your document and sets things up so SCRIPT/VS can figure out what to do about each of the tags. The starter set profile, which is discussed in this book, is called *DSMPROF3*.

In this book, we will tell you how to use the starter set GML tags to mark up your documents.

We will also tell you how to use DSMPROF3 to format your documents.

## *Identifying GML Document Types*

As we mentioned above, the *starter set* of GML tags is designed for *general documents*.

Depending on the kind of organization you work in and the kind of work you do, you might be working on different *specific types* of documents. For example, the documents you work on could include insurance policies, parts catalogs, maintenance manuals, or even functional specifications.

This list could be as long as there are different types of documents that people produce.

[3] This term owes its origins to earlier days in the printing trade, when drafts of manuscripts were literally "marked-up" with specific kinds of instructions to typesetting employees.

Each of these types of documents has its own characteristics and, within your organization, new GML might be developed (by your document administrators and your text programmers)[4] to handle the special requirements for the types of documents you produce.

The starter set supplied by IBM is intended, as its name implies, to give your organization a starting point in developing its own GML. Because we don't know what your specific requirements are, the starter set is designed for *general* rather than *specific* document types.

If you were, for example, working on parts catalogs, your organization might have GML tags for marking up part numbers, which are specific things you need in a parts catalog. You won't find tags of this sort in the starter set. You will find tags for headings, paragraphs, lists, examples, figures, tables, and so on—the *general* kinds of things that are found in most documents, regardless of any specific requirements.

In most cases of designing specific GML markup, the general GML tags in the starter set are used as a core. Therefore, the general rules on markup and most of the tags you will learn about in this book will be valid regardless of the specific markup your organization might develop.

Don't be too concerned about this right now. The purpose of this book is to teach you how to use GML in general and the starter set in particular, and to get you ready to use any specific GML that your organization develops.

## *The Starter Set Profile (DSMPROF3)*

When we talk about the *starter set*, we are referring to the markup (or GML tags) that you put in your document, the profile, DSMPROF3 and the macro library, DSMGML3. When we say *DSMPROF3* does something, we are referring to the IBM-supplied processing instructions (written in the SCRIPT/VS language and contained in the DSMGML3 library) that process that markup. This is an important distinction, because your organization might produce its own processing instructions to process the starter set markup, and the processing results might be different, even though the markup is the same.

For example, DSMPROF3 puts five levels of body heading in the table of contents (head levels 0 through 4). If the style in your organization is to put only four levels in the table of contents, your organization might develop a new profile (or modify DSMPROF3) to do that.

In this book, we teach you how to do markup by illustrating how things look when the IBM-supplied processing instructions process it. But remember that, using GML, you are describing what something is, and not what it will look like on the page.

## *Creating and Formatting Documents*

Using GML and SCRIPT/VS is a two-step process:

1. You create your document (called the *source document* or sometimes the *source file*) using a text editor program or word processing equipment. In this book, we concentrate on *what* you put in the source document (that is, the markup), not how you put it there.

2. You request formatting for your document, at the same time telling SCRIPT/VS what you want done with the formatted output. This could be to:

[4] In this book, the term *document administrator* refers to the person in your organization who is responsible for defining your markup requirements and procedures. The term *text programmer* refers to the person who writes the SCRIPT/VS instructions for processing the markup.

- Display it at your terminal (if you are using one). You could do this to check your formatting at the terminal, or, if you are using a typewriter-like terminal, to print a copy.
- Have it printed on a high-speed printer.
- Have a *camera-ready* copy prepared for final printing.
- Save it as a computer file, which can then be displayed at the terminal or printed out whenever you need a copy.

When you need to update your document, you modify the source document you created in the first step, (again using a text editor program, distributed system or word processor) and then you request formatting again.

## Your Computing System

SCRIPT/VS and GML can be used in many different computer systems. The computer system you use has no effect on the GML markup you put in your source document or on the output results; therefore, most of this book applies regardless of the system you use.

However, the system you use does affect how you request formatting.

If you are using a system called TSO, CMS, or ATMS-III, we will show you in this book how to request formatting. (Note, however, that someone in your organization may have set up some procedures or special commands on your computing system to make it easier for you to request formatting by SCRIPT/VS. If that is the case, you should use your organization's procedures rather than those we show you here.)

If you are using anything other than TSO, CMS, or ATMS-III, you should find out from the document administrator or your supervisor what procedures you should follow to request formatting.

To process SCRIPT/VS documents in batch environments (that is, to have them processed after you have signed off from your computing system), the IBM Document Library Facility program product is required. Processing output can be directed to a printer or to a document data set. Consult the document administrator at your installation for information about the batch processing procedures in use.

# Part One: Tag Guide

In this part, you will find tags for:

# Chapter 2. Getting Started: Paragraphs and Headings

A few of the many GML tags are used in almost every document. Learning these common tags will help you to prepare most documents without having to refer frequently to the GML publications. In this chapter, we'll talk about the tags that are probably used most often—the ones for paragraphs and headings.

Throughout this document, we show tags in capital letters in the text, and in small letters in the examples. You can enter the tags in either uppercase or lowercase in your source document, whichever is easier for you.

## *Paragraphs*

The tag you will use most often is P, for *paragraph.*

Like all tags in the starter set, it begins with a colon (:) and ends with a period (.).[5] So, this paragraph would be entered like this:

```
:p.Like all tags in the starter set, it begins...
```

You *don't* have to skip a line before it, or indent it, or anything like that. It's a good idea to start each sentence within a paragraph on a new line; the lengths of the input lines themselves are not important. Don't end any of the input lines with a hyphen.

**Remember: you don't have to worry about what the paragraph will look like on the page. The starter set will take care of that during formatting.**

We could also have entered the above paragraph like this:

```
:p.
Like all tags in the starter set, it begins...
```

with the tag on a line by itself and the text starting on the very next line.

We will be showing you a lot more paragraphs as we go on; they're all tagged the same way.

Because paragraphs are the *document elements* (or parts of a document) you use most often, you now already know how to mark up half of what you have to enter.

[5] It is not always necessary to end the tag with a period. However, it is *never wrong* to use the period, so using one to end your tag is a good habit to practice! Later, we will discuss in detail when you can omit the period.

## *Headings*

The starter set allows you seven levels of headings, H0 through H6. (Headings will look different, depending on which printer your document is printed.)

H0 (head level 0) is not used too often. You use it when you want to divide a large document into major parts, each part having several chapters. The starter set will always start a new page when it finds an H0.

The heading "Part One: Tag Guide" on page 7 was entered like this:

```
:h0 id=part1.Part One: Tag Guide
```

The "id=part1" is something called an *attribute.* In this case, the attribute is a *labeled* attribute (it contains an attribute name (label), an equal sign, and the attribute's value). Attributes allow us to pass additional information to the processing system along with a tag. In this case, the attribute allows us to make cross-references to the heading from elsewhere in the document. We'll explain all about cross-referencing later in "Chapter 6. Cross-References and Footnotes" on page 63.

Do notice, however, that again we start the markup with a colon (:) and end it (after the attribute) with a period (.).

If we didn't need to make a cross-reference to that heading, we would have entered it like this:

```
:h0.Part One: Tag Guide
```

The important rule to remember about headings is that the text of the heading must be either all on the same line as the end of the markup or all on the next line following the markup.[6]

That is, our heading could have been entered like this:

```
:h0.
Part One: Tag Guide
```

but it *could not* have been entered like this:

```
:h0.Part
One: Tag Guide
```

because the text of the heading is split between lines.

The other important rule about headings is that no tags can be used within the text of the heading. That is, you couldn't put another tag within the line "Part One: Tag Guide".

The H1 (head level 1) tag is used for chapters. You will use it a lot, as you will the rest of the head-level tags.

The starter set always starts a new page when it finds an H1.

---

[6] Actually, the text must be all in the same input record, which can be longer than the length of the line on your terminal. You should find out how to create longer input records using your editor in order to get long heads (and other things that must follow the same rule) into a single record. The maximum length that SCRIPT/VS allows in an input record is 256 characters.

The heading for this chapter, "Chapter 2. Getting Started: Paragraphs and Headings" on page 9, was entered like this:

```
:h1 id=gs.Getting Started:  Creating Paragraphs and Headings
```

Again, ignore the ID attribute. If we didn't want to make a cross-reference to that heading, it would have been entered like this:

```
:h1.Getting Started:  Creating Paragraphs and Headings
```

H0 and H1 have an attribute that the other headings don't have. This is because the starter set uses the text of headings 0 and 1 to print a running footing. If the heading is too long to make a neat running footing, you can specify a shorter version to be used instead. The H0 and H1 tags have an attribute, called STITLE (for *short title*), that allows you to specify the shorter version to be used for the running footing.

You would specify it like this:

```
:h1 stitle='Short Footing Title'.Very Very Long Chapter Title
```

Notice that the text you want used for the running footing is the value that you give to the STITLE attribute, and that it goes in single quotation marks. Many attributes are specified this way; that is, with the attribute name, followed by an equal sign, followed by the value in single quotation marks.[7] You'll see as we go along that there are other kinds of attributes and we'll show you how to specify them.

As you can see, we still managed to get the whole thing on one line. If your short title and your full title are too long to fit on one line, you can enter the text of the heading on the line following the tag and attribute, like this:

```
:h1 stitle='Short Footing Title'.
Very Very Long Chapter Title
```

If you have a lot of attributes to specify on a tag, you could spread the markup itself over several lines, as long as you don't split an attribute in the middle. (You couldn't put "stitle= " on one line and "Short Footing Title" on the next.) And remember that the text of the heading must be either all on the same line as the end of the markup or all on the next line. You could, for example, enter a level 1 heading with two attributes like this:

```
:h1 stitle='Short Footing Title'
id=part1.
Very Very Long Chapter Title
```

You'll notice that the period (.) was entered after the last attribute, before the text associated with the heading. The period indicates that the markup is ended—what follows is the text. (The official name of that period is the *markup-content separator.*)

**Note:** The ampersand (&) character should not be used in the STITLE text of the :H0 or the :H1 tag. Also, when the STITLE attribute is not specified, the ampersand should not be used in the text of the section heading. The ampersand is the page number symbol character and its use in a :H0 or :H1 tag will result in the current page number being printed in the text of the running footing. If an ampersand must be used in the text of a head level control, use the &amp symbol.

---

You only need to include the quotation marks if the value of the attribute contains blanks or special characters, such as punctuation characters. Here we need them because we have blanks in our attribute value. However, the single quotation marks around an attribute value, like the period at the end of the markup, are *never* wrong, so when in doubt, put them in.

## *Example of a Head 2*

This heading was entered like this:

```
:h2.Example of a Head 2
:p.This heading was entered....
```

The starter set puts headings in the proper heading format for each heading level, no matter how you enter them at your terminal. The proper heading format may be all capitals or it may be both uppercase *and* lowercase characters, depending on the output device you use to print your document. You should, however, enter your headings as you want them to appear in the table of contents because they are put in the table of contents in the same form you enter them. (Take a look at the table of contents for this book.)

### Example of a Head 3

This one was entered like this:

```
:h3.Example of a Head 3
:p.This one was entered....
```

#### *And an Example of Head 4*

Like so:

```
:h4.And an Example of Head 4
:p.Like so:
```

Head level 4 is the last level that the starter set will put in the table of contents. Heads 5 and 6 are not included in the table of contents.

**Example of Head 5:** This was entered like this:

```
:h5.Example of Head 5
:p.This was entered... .
```

***And a Head 6:*** This one was entered like this:

```
:h6.And a Head 6
:p.This one was... .
```

## More About Heads

Did you notice that we didn't enter a colon following the level 5 or the level 6 headings above? The starter set puts it there automatically, but *only* when a P tag immediately follows the H5 or H6 tag line. If you don't have a P tag immediately after the H5 or H6 tag line you will not get the colon. Furthermore, if your heading text is too long or if other tags follow the heading tag, the colon may end up in an unpredictable location.

You may want the headings in your document numbered. ***You do not have to number the headings yourself.*** The starter set will do it for you if you ask it to when you request formatting. We'll discuss how to get the formatting you want and how to get headings numbered when we get to "Chapter 12. Things You Can Specify At Run Time" on page 115.

## Exercise: Paragraphs and Headings

So now you know how to enter paragraphs and headings, which means you can enter sixty percent of what you have to do.

See if you can create a document (call it EXER1) that will look like this: (See "Markup for Getting Started—Paragraphs and Headings" on page 193 for the correct markup.)

---

### This is a Head Three

It has two short paragraphs following it. This is the first paragraph. Each sentence was started on a new line.

This is the second paragraph. Look to see how DSMPROF3 has formatted it (a skip before, and no indention).

**This is a Head Five:** The paragraph following it begins on the same line as the head, even though it was entered on a separate line.

The next paragraph under the head five formats like this.

---

To see the results back at your terminal (in single column rather than the *offset style* we show you here), do one of the following.

**In TSO enter:** [8]

```
  script exer1 prof(dsmprof3) cont
```

EXER1 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**In CMS enter:** [8]

```
  script exer1 (prof(dsmprof3) cont
```

EXER1 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**In ATMS-III enter:** [8]

```
  script * prof(dsmprof3) cont
```

In ATMS-III, the document to be formatted (EXER1—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

**Note:** When the starter set formats back to a display terminal, it lays out the page as if the terminal were a typewriter-like terminal, so you may only see part of a full page at a time.

The CONT (continue) option at the end of each command allows processing to continue after SCRIPT/VS detects an error condition and issues an error message. If you do not include this option, formatting stops at the first error. When SCRIPT/VS encounters an error that is too severe for processing to continue, it stops processing even when CONT is specified.

[8] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

# Chapter 3. Creating Lists

We'll discuss here the various kinds of lists you can create with the starter set. The starter set takes care of all the formatting for lists.

## *Simple, Unordered, and Ordered Lists*

Simple lists are just what you think they are. For example, here's a simple list:

bread

butter

cheese

bananas

You can also have a *compact* simple list. A compact list leaves out the blank spaces between the items of the list:

pears
milk
lettuce
parmesan
oregano
paper towels
whipping cream
lemons
lamb chops

Unordered lists are similar to simple lists except that each item in an unordered list is preceded by a special symbol. (The special symbol used depends on the device on which you are having your output printed.) You would use an unordered list when the items in the list are fairly long, maybe even many paragraphs, but you don't need them in any particular sequence.

Here's an example of an unordered list:

- This is an item in an unordered list. To distinguish it from other items in the list, the starter set puts a bullet beside it.

  This item consists of two paragraphs. This paragraph does not get a bullet because it is not a separate list item.

- This is a separate list item in our unordered list.

And then there's the ordered list. Use an ordered list when the items you're listing need to be in a specific order. An example of an ordered list follows.

1. Cream butter and sugar together until fluffy.
2. Beat in egg yolks one at a time.
3. Add nutmeg, cinnamon, and vanilla, and mix thoroughly.
4. Add flour and beat for five minutes. The batter should be smooth and glossy and stream off the spoon in ribbons.
5. Fold in beaten egg whites.

   Do not overmix; the batter should be light and fluffy.

When you are doing an ordered list, you don't have to number the items yourself. The starter set will do it for you. This will save you a lot of work when you decide to insert or delete items, or rearrange them.

Unordered and ordered lists can also be compact.

Each of these list types starts with a tag that tells what kind of list it is and ends with a matching end tag.

This is our first case of end tags. End tags are used when it is necessary to indicate that something has ended. (You don't need to end a paragraph because the start of a new paragraph shows that you're finished with the old one. And you don't need to end a heading because it all goes on one line and nothing else is allowed on that line.)

End tags consist of the tag name, prefixed with the end tag delimiter. The end tag delimiter in the starter set is the two characters ":e"—so, if the tag for simple list is ":sl" (which it happens to be), then you know the tag for ending the list is ":esl."[9]

[9] It is important to understand that the end tag is "sl" prefixed with the end tag delimiter (":e") rather than "esl." This is because your organization could change the end tag delimiter (as they could also change the tag delimiter). You might, for example, have an exclamation point as the tag delimiter and two exclamation points as the end tag delimiter, so your simple list tags would actually be entered as "!sl" and "!!sl."

This is why in this book we say *matching end tag* or *SL end tag* rather than "ESL"—to remind you that the "E" is part of the *delimiter* and not the tag itself.

Here are the tags for starting and ending the lists we've been talking about:

| **Tags** | **List Type** |
|---|---|
| **:sl, :esl** | Simple list |
| **:ul, :eul** | Unordered list |
| **:ol, :col** | Ordered list |

**Note:** ***Regardless of the list type***, all items within these lists are tagged with the same tag—LI (for *list item*). This saves having to remember a lot of different tag names.

Like headings, list items also can have ID attributes. Any list item that you give a *name* by using the ID attribute can be referred to from some other place within your document.

Even though any list item can be *named*, you will probably find naming useful only within ordered lists where you can refer back to a particular number. (We'll talk about this in more detail in "Chapter 6. Cross-References and Footnotes" on page 63.)

List items do not need an end tag because the list item is ended by another list item or by the end of the list itself.

So, here's how you would enter the simple list we show above:

```
:sl.
:li.bread
:li.butter
:li.cheese
:li.bananas
:esl.
```

We began the list with SL, marked every item in it with LI, and ended it with the SL end tag.

Here's a case (the SL tag) where the period following the markup is optional. Because the SL tag is always followed by another tag (LI), it is not necessary to end the SL markup with a period. (The official name of that period is the *markup-content separator*.) Because the SL tag has no text of its own, the period is not required in this case. However, it is not wrong to put the period in. If you get into the habit of *always* using the period, then you don't have to worry about those cases where leaving it out causes an error.

If you want to make a list compact, add the COMPACT attribute to the SL tag. The markup for the simple, compact list we showed you earlier would look like this:

```
:sl compact.
:li.pears
:li.milk
:li.lettuce
:li.parmesan
:li.oregano
:li.paper towels
:li.whipping cream
:li.lemons
:li.lamb chops
:esl.
```

Notice that the COMPACT attribute is different from the attributes we've already looked at; it's just the single word "COMPACT." This form of attribute is called a *value attribute*, because you enter only the value and not an attribute label.

Value attributes (single-word attributes) are *not* enclosed in single quotation marks.

Here's how we marked up our unordered list:

```
:ul.
:li.This is an item in an unordered list.
To distinguish it from other items
in the list, the starter set
puts a bullet beside it.
:p.This item consists of two paragraphs.
This paragraph does not get a bullet
because it is not a separate list item.
:li.This is a separate list item
in our unordered list.
:eul.
```

Notice that the second paragraph in the first item is marked with the P tag. The starter set knows what to do with that second paragraph because it's within a list item.

If we had wanted this list to be compact, we would have put the COMPACT attribute on the UL tag, just as we did with SL. The COMPACT attribute applies only to the space between list items, and not to any space between paragraphs within a list item, so you would probably only want to use it on lists with short list items.

And here is the ordered list:

```
:ol.
:li.Cream butter and sugar together until fluffy.
:li.Beat in egg yolks one at a time.
:li.Add nutmeg, cinnamon, and vanilla,
and mix thoroughly.
:li.Add flour and beat for five minutes.
The batter should be smooth and glossy
and stream off the spoon in ribbons.
:li.Fold in beaten egg whites.
:p.Do not overmix; the batter
should be light and fluffy.
:eol.
```

Again, we could have used the COMPACT attribute, just as we did with our simple list.

## Creating Definition Lists

Definition lists are used when you want to pair a term or phrase with its description.

Here's an example of a definition list with headings:

| *TERM* | *DESCRIPTION* |
|---|---|
| **gopher** | A burrowing rodent that feeds on roots of plants. |
| **lawn** | Gopher highway.<br><br>Can be identified by dinner-plate-sized mounds of dirt where grass used to be. |
| **agapanthus** | Lovely flowering plant, the roots of which are the preferred food of gophers.<br><br>If your flourishing agapanthus suddenly keels over, it means a gopher has had a feast. |

This list is different from the other lists in several ways. This is a list that can have headings; it also has three labeled attributes and two value attributes.

To mark up a list like this, you must first decide how much space you want reserved for the term and the term heading (the left hand side). If it's 10 character spaces, you don't have to specify anything, because 10 character spaces is the *default*. (A default is the automatic value of an attribute if you don't assign it a value yourself.) For the list above we specified 22 millimeters, to give us enough space for the term "agapanthus" as well as additional space for the *gutter* (the space between the terms and their descriptions).

**Note:** The default unit of measure is actually *character spaces*. But to provide a more universal example, we used millimeters. For more information about space units in general, and character spaces in particular, see "Space Units" on page 34.

You must also decide whether or not you want the headings and terms to be highlighted. Highlighting the terms and headings is the default, which we chose to do here. The appearance of highlighting depends on the device you are printing on. Highlighting is explained in "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71.

The starting tag for a definition list is DL and, like the other lists, it has a matching end tag. DL has five attributes:

| | |
|---|---|
| **BREAK** | for specifying that the description should start on the next line after the term if the length of the term exceeds TSIZE. In cases where the term is longer than TSIZE, the default is to leave one blank after the term and then start the definition. |
| **COMPACT** | for specifying when you don't want blank lines between the individual terms and descriptions. The default is to put blank lines between the individual terms and descriptions. |
| **TSIZE** | for specifying the combined space for the definition terms and the *gutter* (space) between the terms and headings. The default for TSIZE is 10 character spaces. This will be discussed in more detail in "Space Units" on page 34. |

**HEADHI** for specifying the highlighting for the headings. The default for HEADHI is highlight level 3. This will be discussed in more detail in "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71.

**TERMHI** for specifying the highlighting for the terms. The default for TERMHI is highlight level 2. This will be discussed in more detail in "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71.

The defaults for each of the attributes is used unless you specifically ask for something else when you enter your markup.

Within the definition list, you tag the heading for the terms with the DTHD (definition term heading) tag and the heading for the descriptions with the DDHD (definition description heading) tag. Similarly, you tag the terms with the DT (definition term) tag and their accompanying descriptions with the DD (definition description) tag.

So a definition list is made up of headings for the terms and descriptions (DTHD and DDHD) and pairs of DT and DD tags, within DL and its matching end tag.

Here's the markup for our example on page 19:

```
:dl tsize=12.
:dthd.TERM
:ddhd.DESCRIPTION
:dt.gopher
:dd.A burrowing rodent that feeds on roots
of plants.
:dt.lawn
:dd.Gopher highway.
:p.Can be identified by dinner-plate-sized
mounds of dirt where grass used to be.
:dt.agapanthus
:dd.Lovely flowering plant,
the roots of which are the
preferred food of gophers.
:p.If your flourishing
agapanthus suddenly keels over,
it means a gopher has had a
feast.
:edl.
```

There's the P tag again. Because it is within a definition description, the starter set knows exactly what to do with it.

Notice also that we did not have to put the value of the TSIZE attribute in single quotation marks. The single quotation marks are only required when there are special characters or blanks in the attribute value. (For example, if you had a period in the attribute value, SCRIPT/VS would think it was the period meaning the end of the markup if you did not put the value in quotation marks. Similarly, blanks and other special characters, such as punctuation characters, could be misinterpreted by SCRIPT/VS unless you use the quotation marks.) The use of the quotation marks is never wrong on labeled attributes (attributes with an equal sign), so when in doubt, put them in.

The main rule you need to remember about definition lists is that the text for the definition term, definition term heading, and definition description heading must be either all on the same line as the tag or all on the next line, and there can't be any tags within the text. (This is the same rule you've already learned for headings.)

If we use the same markup shown above and use several different attributes on the DL tag, we can change the appearance of the listing.

For example, by changing only the first line of our markup for the definition list to look like this:

```
:dl tsize='8' headhi='1' termhi='0' break.
```

the output would then look like this:

---

*TERM* *DESCRIPTION*

gopher A burrowing rodent that feeds on roots of plants.

lawn Gopher highway.

Can be identified by dinner-plate-sized mounds of dirt where grass used to be.

agapanthus
Lovely flowering plant, the roots of which are the preferred food of gophers.

If your flourishing agapanthus suddenly keels over, it means a gopher has had a feast.

---

As you can see in the above example, specifying values for the HEADHI and TERMHI attributes changes the appearance of the definition list headings and terms from the defaults. See "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71 for information on highlighting.

## *Glossary Lists*

There is an additional specialized type of list for which the starter set has tags—a glossary list.

Here is an example of a couple of items that might appear in a glossary: (In fact, these appear in the glossary at the back of this book.):

---

**formatter:** A computer program that prepares a source document to be printed.

**Generalized Markup Language (GML):** A language for describing the characteristics of a document without respect to particular processing.

**GML:** Generalized Markup Language

---

The glossary usually appears in the back matter of a book. Take a look at the "Glossary" on page 203 for an example of what this book's glossary looks like.

The glossary term is printed in highlight level 2 followed by a colon.

**Note:** The appearance of highlighting depends on the device you are printing on. For more information on highlighting see "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71.

The starting tag for a glossary list is GL. You end the glossary list with its matching end tag.

You can use the COMPACT attribute with glossary lists, but it probably wouldn't be a good idea, because it would look too crowded.

The other attribute that you can specify on the GL tag is TERMHI. It works the same on glossary lists as it does on definition lists—the default is to highlight the term. (We'll discuss highlighting in "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71.)

Here is the markup for the glossary list example:

```
:gl.
:gt.formatter
:gd.A computer program that prepares
a source document to be printed.
:gt.Generalized Markup Language (GML)
:gd.A language for describing the characteristics
of a document without respect to particular processing.
:gt.GML
:gd.Generalized Markup Language
:egl.
```

The text for the glossary term must be either all on the same line as the tag or all on the next line. There can't be any tags within the text.

## *Nesting Lists*

A nested list is a list that is contained within another list. The starter set allows all of the lists we've just described to be nested within another list of the same or different type, many levels deep. Because each nested list is indented from the left edge of the previously nested list, the only restriction is the width of your column.

Perhaps the most common case of nested lists is ordered lists, such as the example which follows.

1. First item in the first level list.
2. Second item in the first level list. It has a nested list within it.
    a. First item in the second level list.
    b. Second item in the second level list.
3. Third item in the first level list.

If you remember that each list must start and end with a tag that says what type of list it is, and that all list items within a list are marked as list items, and that you don't have to worry about numbering the items, you won't have any trouble nesting lists.

This is how the nested list was marked up:

```
:ol.
:li.First item in the first level list.
:li.Second item in the first level list.
It has a nested list within it.
:ol.
:li.First item in the second level list.
:li.Second item in the second level list.
:eol.
:li.Third item in the first level list.
:eol.
```

The lines over on the right are to help you see where the lists begin and end; they're not part of the markup.

As you can see, the second list is entirely contained within a list item of the first list. Similarly, when you are nesting lists within a definition list, the nested list must be entirely contained within a definition description of the definition list.

The same general approach is used for nesting lists of different types. Here, for example, is a case of an unordered list with an ordered list within it, and a definition list within that:

- When creating a CMS file, you must specify these things:
  1. filename
  2. filetype

     In System Information, we generally use these file types:

     | | |
     |---|---|
     | *SCRIPT* | For files containing text.<br><br>SCRIPT/VS assumes a filetype of SCRIPT and thus you do not have to specify it when you request formatting. |
     | *EXEC* | For files containing CMS commands. |
     | *OPTIONS* | For files containing the options to be used when SCRIPT/VS formatting is requested. |

  3. filemode

     Filemode defaults to A1 if not specified.
- When creating a TSO file...

Here's the markup for this example (and again, we've used lines to show you where the individual lists begin and end):

```
:ul.
:li.When creating a CMS file,
you must specify these things:
:ol.
:li.filename
:li.filetype
:p.In System Information, we
generally use these file types:
:dl termhi=1.
:dt.SCRIPT
:dd.For files containing text.
:p.SCRIPT/VS assumes a filetype of
SCRIPT and thus you do not have to
specify it when you request formatting.
:dt.EXEC
:dd.For files containing CMS commands.
:dt.OPTIONS
:dd.For files containing the options
to be used when SCRIPT/VS
formatting is requested.
:edl.
:li.filemode
:p.Filemode defaults to A1
if not specified.
:eol.
:li.When creating a TSO file...
:eul.
```

If you run into trouble with nested lists, it is probably because you haven't ended your lists properly. Check to make sure that each list is ended with a matching ending tag.

## *List Parts*

Sometimes in an ordered list you want to break the list for some explanatory material and then resume the numbering where you left off. The starter set lets you do this with the LP (list part) tag.

For example, suppose you wanted to do this:

1. Saute the shallots and chopped mushrooms until the shallots are tender and the liquid from the mushrooms has cooked away.
2. Brown the sausage and add to the mushroom mixture.

The above may be prepared several hours in advance and refrigerated. Then, 30 minutes before serving time, finish the dish.

3. Mix one can of tomato sauce with the mushroom and sausage mixture and bring to a slow simmer.
4. Add the heavy cream and immediately pour into a casserole.
5. Pop into 350 degree oven for 15 minutes.

We used the LP tag to break the list. It was then continued again by the next LI (list item) tag.

If you have more than one list part between items in a list, begin each list part with the LP tag.

Here's the markup:

```
:ol.
:li.Saute the shallots and chopped
mushrooms until the shallots are
tender and the liquid from the
mushrooms has cooked away.
:li.Brown the sausage and add to
the mushroom mixture.
:lp.The above may be prepared several
hours in advance and refrigerated.
Then, 30 minutes before serving time,
finish the dish.
:li.Mix one can of tomato sauce
with the mushroom and sausage mixture
and bring to a slow simmer.
:li.Add the heavy cream and immediately
pour into a casserole.
:li.Pop into 350 degree oven for 15 minutes.
:eol.
```

List parts can be used in any of the list types.

## *Implied Paragraphs*

We mentioned a little earlier that all paragraphs were tagged with a P. That's not entirely true. You may have noticed when we talked about list items and definition descriptions that we said that any paragraphs after the first one were tagged with a P. Of course, the first one is also a paragraph, but since we've already described it as a list item (with the LI tag) or a definition description (with the DD tag), we don't have to also say that it is a paragraph, because that is "implied" by the LI or DD.

Implied paragraphs show up in a number of places; you'll be able to see from the examples where you need an explicit P tag and where you don't.

## *Exercise: Lists*

See if you can create a document that looks like this (call it EXER2); you'll find the markup in "Markup for Lists" on page 194:

Please send the following items:

| *Number* | *Description* |
|---|---|
| **A107C2** | Class A Widget, 100 x 200 mm in assorted colors (no white). |
| **B321DJ** | Brown and yellow Whuzzit, with linkage for optional nutcracker attachment. |
| **BY7532** | Assorted gimcracks, with the following characteristics: |
| | • Soft |
| | • Cuddly. |

When I get around to it, I will

1. Balance my checkbook.
2. Prune the shrubbery.
3. Write to the following:

   My mother in New Rochelle  
   My great aunt in Detroit  
   My niece in Ossining  
   My friend in South Bend.
4. Clean out the garage.

To see the results back at your terminal (in single column), do one of the following:

**TSO users enter:**[10]

```
script exer2 prof(dsmprof3) cont
```

EXER2 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**CMS users enter:**[10]

```
script exer2 (prof(dsmprof3) cont
```

EXER2 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[10]

```
script * prof(dsmprof3) cont
```

The document to be formatted (EXER2—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

[10] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

# Chapter 4. Creating Examples and Figures

General documents often have many examples and figures. We'll show you how to do those next.

## *Examples*

Examples are easy; you just begin them with an XMP tag and end them with an XMP end tag.

Within the range of the XMP tag, the starter set turns off formatting and treats each input line as an output line; it does not link them together into paragraphs. The starter set keeps all of the lines of an example together, so an example cannot be longer than the column. If an example will not fit within the remaining space of a column, the example will start at the beginning of the next column.

Here's a typical example:

```
10 LET A = B
20 IF A GT C THEN GO TO 40
30 LET A = C
40 PRINT A, C
```

The markup looks like this:

```
:xmp.
10 LET A = B
20 IF A GT C THEN GO TO 40
30 LET A = C
40 PRINT A, C
:exmp.
```

XMP has a DEPTH attribute, which allows you to specify that you want the starter set to leave space for you to paste in the example. The vertical space is specified in *vertical space units,* which can be any of the ones described in "Space Units" on page 34. If you specify the DEPTH attribute and also put some text in your figure, the starter set will give you the vertical space first, and then put in your text. The DEPTH attribute works the same way for examples as it does for figures, so we describe it in more detail in "Figures" on page 28, where it is used more often.

## Paragraph Continuation

Sometimes, following an example or a list, you may want to continue a paragraph that was begun before the example or list.

You do this by using the paragraph continuation tag (PC) instead of the P tag following the example or list. For instance, suppose you had something like this:

When you enter

```
loc /A + B
```

the editor will locate the next line containing A + B.

Your markup would look like this:

```
:p.When you enter
:xmp.
loc /A + B
:exmp.
:pc.the editor will locate the next line
containing A + B.
```

Since there is no apparent difference in how the starter set formats a paragraph and a paragraph continuation, you may wonder why you have to bother to specify them differently.

Here's why:

The formatting style the starter set uses is called *block paragraph*, where the first line of the paragraph is not indented. However, if your organization were to change the formatting style to indent paragraphs, you would not want your paragraph continuations to also be indented.

By correctly specifying the markup at the outset, you never have to worry about a change in formatting style producing unexpected output.

## Figures

Figures are a little more complicated than examples, but only because figures have more attributes that let you do more things.

For instance, you can choose to have a figure formatted within the column (if you're formatting for two columns or the offset style we are using in this publication) or formatted the full width of the page. And you can either have it placed exactly where you entered it (inline), or you can allow the starter set to move it so as to fill out a page. (The figure *floats* (moves) to the top or bottom—you can say which—of the next column or page.)

You can also choose to have a figure set off with rules across the page, put in a box, or formatted with no frame at all.

You can tell the starter set you want space left in the figure so you can paste in a picture.

You can give the figure a caption, which will also cause it to be listed in the list of illustrations (and we'll tell you about that later, too). You can extend the figure caption with a figure description, too. (Only the caption itself goes in the list of illustrations.)

If you give your figure a name (ID) and a caption, you can make cross-references to that figure. (We'll cover cross-references in the next chapter.)

Needless to say, because you can do all of these things, the markup for a figure is a little more extensive than anything you've seen before. But, all you have to do is figure out what you want and then ask for it.

Here's the markup for an inline figure (no floating) that is the width of the column. We've let the frame default to rules. In this case, your markup would look like this:

```
:fig place='inline' width='column'.
Just as with XMP,
the input lines entered in the
figure body are treated
as output lines;
they won¬t be formatted.
:efig.
```

The figure is begun with a FIG tag with PLACE and WIDTH attributes and ended with a FIG end tag. We've made this figure inline and haven't put a caption on it. The result would look like this (notice that, like examples, the text is printed as it is entered):

```
Just as with XMP,
the input lines entered in the
figure body are treated
as output lines;
they won't be formatted.
```

Don't be confused by the double set of lines; the lines on the outside are the ones we use in this book to distinguish our examples of output from our regular text. We do this rather than use the XMP tag because most of the examples in this book are *live* examples (the exceptions are in the indexing chapter). That is, the examples are marked up exactly as we show you and we let the starter set produce the results just as it would in your document.

The inside lines are the ones the starter set produces when the FIG tag defaults to the FRAME attribute, which we'll tell you about below. They are called *rules*.

Let's take a look at the FIG tag itself. FIG has five attributes, so a fully specified FIG would look like this:

```
:FIG ID=name
     PLACE=placement
     WIDTH=horizontal space
     FRAME=frame
     DEPTH=vertical space.
```

where:

**ID** = name

is the attribute that allows you to make cross-references to a figure (when you also use the FIGCAP tag). We describe cross-referencing in "Chapter 6. Cross-References and Footnotes" on page 63.

**PLACE** = placement

is how you specify where you want the figure to go. You can specify any one of three placements:

| | |
|---|---|
| **TOP** | This will cause your figure to float to the top of the next available column or page. If necessary, text will be brought up in front of the figure to fill out the current column or page. *TOP* is the default. |
| **INLINE** | This will cause your figure to be set exactly where you entered it; no text will be moved around it. |
| **BOTTOM** | This will cause your figure to float to the bottom of the next available column or page. If necessary, text will be brought up in front of the figure to fill out the column or page. |

**Note:** Usually, you will want a floating figure to have a caption and a cross-reference to it, sometimes called a *figure callout.*

**WIDTH** = horizontal space

is how you specify the width of your figure. It can be one of three things:

| | |
|---|---|
| **space units** | This can be specified as any of the horizontal space units described in "Space Units" on page 34.<br><br>If the width you specify for your figure is narrower than the width of the current column, the starter set will set the figure in the column. If the width you specify for your figure is wider than the width of the current column, the figure will be set the full page width.<br><br>Specifying space units rather than PAGE or COLUMN allows you to create figures that are somewhat independent of the general layout of the document and gives you a flexibility that COLUMN or PAGE do not allow. Then, if you print the document with different column or page widths than originally specified, or if you decide to change a single-column document to double column, you will not have to change the markup for your figure.<br><br>For these reasons, it is a good idea to use space units in specifying your figure width rather than COLUMN. |
| **COLUMN** | This sets the figure to the width of the current column.<br><br>Space units should be explicitly used where there is any possibility that the column width might change at some future point. |
| **PAGE** | This sets the figure width to the full page width. *PAGE* is the default. |

**FRAME** = frame

tells how you want the figure framed. You can ask for any one of four things:

| | |
|---|---|
| **RULE** | gives you the rules we showed you in the example above. *RULE* is the default. |
| **BOX** | gives you a box around the figure. |
| **NONE** | says "don't put anything around this figure." |
| **'string'** | allows you to specify a string of characters to be used in place of the rules. For instance, if you said `frame='*'` you'd get a line of asterisks instead of a rule. |

**Note:** When a figure frame defaults to rules or a specific *string* is requested, the string or rule appears across the top and bottom of the figure when the figure is placed *inline*. When a figure is floated to the top of a page, only the rule or string at the bottom of the figure appears. Similarly, when a figure is floated to the bottom of a page, only the rule or string at the top of the figure appears.

**DEPTH** = vertical space

specifies that you want vertical space left for something to be pasted in. The vertical space is specified in *vertical space units*, which can be any of the ones described in "Space Units" on page 34.

If you specify the DEPTH attribute and also put some text in your figure, the starter set will give you the vertical space first, and then put in your text.

The depth attribute can be used on the XMP tag, with the same result.

The FIG tag and its attributes are entered before you enter the contents of your figure. Once the figure content has been completed, you can decide if you want a figure caption.

If you want a figure caption (and you must have one to get the figure numbered and placed in the list of illustrations), use the FIGCAP tag at the end of the body of your figure, before the FIG end tag. The text of the caption must go on the same line as the FIGCAP tag or entirely on the next line; it must not contain any tags.

*You don't have to enter the word "Figure" or the figure number.* The starter set takes care of all that for you.

If you have a lot that you want to say about the figure, you can also have a figure description, which can go on for as many lines as you need. It won't be put in the list of illustrations, but it will be added to the figure caption in the text.

You do this with the FIGDESC (figure description) tag, which is placed immediately after the FIGCAP tag, before the FIG end tag.

To summarize, the information used to identify a starter set figure can include both a figure caption (FIGCAP) and a figure description (FIGDESC).

The FIGCAP tag:

- Numbers the figure automatically
- Formats the text that was included with the FIGCAP tag
- Places the figure number and text from the FIGCAP tag into the list of illustrations.

The FIGDESC tag:

- Enters a colon (:) after the text of the FIGCAP
- Follows the colon with the text of the figure description (FIGDESC) tag.

Now we'll show you a couple of examples of what we've been talking about.

This is an inline figure with
a box around it.
Because we said it was 16
picas wide,
the starter set
can decide whether it can
fit in the column or must
be full-page width.

**Figure 1. Example of an Inline Figure:** This example also has a figure description associated with it, which can be several lines.

Here is the markup for the previous figure:

```
:fig width=16p place=inline frame=box.
This is an inline figure with
a box around it.
Because we said it was 16
picas wide,
the starter set
can decide whether it can
fit in the column or must
be full-page width.
:figcap.Example of an Inline Figure
:figdesc.This example also has a figure
description associated with it, which can be several lines.
:efig.
```

Notice that you do not have to worry about the punctuation between the figure caption and the figure description. The starter set supplies the punctuation when it sees the FIGDESC tag.

At this point, you might also want to take a look at the list of illustrations (following the table of contents) to see what the starter set has done with the figure caption there.

Here's an example of a full page figure with vertical space left in it. We'll give this one a frame composed of "*-" by specifying "*-" as the value of the FRAME attribute on the FIG tag.

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

**Figure 2. Page-Wide Figure with Space**

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-

Here is the markup for the above figure:

```
:fig place='inline' depth='10p' width='page' frame='*-'.
:figcap.Page-Wide Figure with Space
:efig.
```

The four attributes we've specified with this figure tag aren't in any particular order. The order in which the attributes are placed is not important on any of the tags.

In this chapter, we've made all of our figures inline to show examples with their explanations and mark up. However, this is not what you usually want to do with figures, because if you make a figure inline and there isn't enough space for it in the column, it will leave the rest of that column empty and go to the next column or page to fit in the figure. This can leave big areas of white space in your document where you don't really want them. As a rule, all but the very smallest of figures should be allowed to float (this is why floating to the top is the default on the PLACE attribute).

## *Including Page Segments*

You can include page segments that may be a picture, logo, or some other kind of image when printing on a page device. In order to integrate that image into your text, you need to specify the SCRIPT/VS .SI [Segment Include] control word.

The .SI [Segment Include] control word identifies the place in your document where you want that image placed. If you want the included segment to have a caption, you can specify that the page segment be included within the FIG start and end tags, like this:

```
:fig place=inline.
.si segname
:figcap.The Golden Fleece
:efig.
```

where:

"segname" is the external name of the file that contains the image to be imbedded.

If the document will be printed on an MVS, an ATMS-III, or a TSO system, **segname** must be the name of a member in a segment library. However, on a CMS system, **segname** must be a CMS file name.

Segments are not supported in a VSE system.

The image contained in the file named "segname" will be included in the document when it is printed on the appropriate page device, provided the page segment is available.

If the page segment you request is in the default library, you don't need to specify the library that the page segment is in. If, however, the page segment is in a segment library you have created, you must specify the SEGLIB option on the SCRIPT command. SCRIPT/VS searches only one segment library, either the default library or the one that you created.

## Reserving Space for a Page Segment

Sometimes, when you are working on a draft of a document for example, you may want to reserve space for a page segment that is incomplete or that does not yet exist. You can reserve this space by specifying a width, a depth, or both. As an example, if your document were one-column and you expected the page segment to take up a large part of the output page, you might specify:

```
  .si clash width 6i depth 7.5i
```

which reserves 6 inches of horizontal space and 7.5 inches of vertical space for the proposed page segment. Later, when the actual page segment is included in your document, any width and depth values you have specified will be replaced by the actual size of the page segment as it has been specified in the segment library.

See "Instructions to SCRIPT/VS" on page 119 for descriptions of the SEGLIB and NOSEGLIB options and the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for more information on the .SI [Segment Include] control word.

## *Space Units*

Space units supported by SCRIPT/VS are:

inch
millimeter
centimeter
pica
cicero
line (vertical space unit)
character (horizontal space unit)
em-space
device units.

The space units can be entered in either uppercase or lowercase. Fractional numbers can be entered with either a period (.) or a comma (,) so that 1.25 and 1,25 are both correct.

**Note:** If you enter an attribute value using a **period** as a decimal point, you must also enclose that value in single quotation marks; otherwise SCRIPT/VS will interpret that period as the end of your markup. Here's how such a markup should look:

```
depth='3.2i'
```

Here are explanations of the various space units and how you specify them:

**inch**

There are 39.37 inches in a meter.

Specify inches as a decimal number (which can have a decimal point or comma and up to two decimal places), followed by an "I," like so:

| | |
|---|---|
| 3i | (3 inches) |
| 2.5i | (2 1/2 inches) |
| 4,25i | (4 1/4 inches) |

**millimeter**

One thousandth of a meter. There are 25.4 millimeters in an inch.

Specify millimeters, like inches, as a decimal number (which can have a decimal point or comma and up to two decimal places) followed by "MM," like so:

| | |
|---|---|
| 85mm | (85 millimeters) |
| 40.25mm | (40 1/4 millimeters) |
| 15,5mm | (15 1/2 millimeters) |

**centimeter**

One hundredth of a meter. There are 2.54 centimeters in an inch.

Specify centimeters just like millimeters, except use "CM" instead of "MM," to look like this:

| | |
|---|---|
| 1cm | (1 centimeter) |
| 2.5cm | (2 1/2 centimeters) |
| 18,25cm | (18 1/4 centimeters) |

**pica**

A standard printer's measurement in the United States and Great Britain. A pica is .1663 inches, and there are twelve points to a pica (roughly, 6 picas to an inch and 72 points to an inch).

Specify the number of picas, followed by a "P," followed by a number of points, like so:

| | |
|---|---|
| 3p2 | (3 picas and 2 points) |
| 12p | (12 picas, no points) |
| p8 | (no picas, 8 points) |

Picas can be specified in tenths of units (for example, 1.5p = 1.5 picas).

**cicero** A standard printer's measurement in most countries except for the United States and Great Britain. Ciceros (and didot points) are comparable to picas and points. The cicero is .1776 inches, and there are twelve didot points in a cicero.

Specify the number of ciceros, followed by a "C," followed by the number of didot points, like so:

| | |
|---|---|
| 4c3 | (4 ciceros and 3 didot points) |
| 5c | (5 ciceros, no points) |
| c8 | (no ciceros, 8 didot points) |

Ciceros can be specified in tenths of units (for example, 1.5c = 1.5 ciceros).

**lines** A number of output lines, specified as just a number, like so:

| | |
|---|---|
| 4 | (lines) |

which would leave 4 blank lines in the output.

For page devices, line spaces are equal in size to the line spacing of the current font.

As a general rule, if you are leaving space for a figure to be pasted in or included at the time of printing, you would not want to use lines as your measure. The space left then depends on the size of the font you are using for page devices, or the number of lines per inch for line printers. Because this could change when you print on different devices or even on the same device with a different lines-per-inch specified, the space left for your figure could change and the figure wouldn't fit. Therefore, you are better off if you use one of the other measures, such as inches or picas, on your DEPTH attribute.

**characters** A number of output characters, also specified as just a number, like so:

| | |
|---|---|
| 12 | (characters) |

which specifies a width equal to 12 character spaces in the default font.

For page devices, character spaces are equal in size to the figure space (approximately one-half an em-space) of the default font. The default font is the initial font at the start of the formatting run. For all other devices, a character width is the width of a blank.

You would use this specification when the width you specify is keyed to the character count. (If you are formatting on a page device, use ems instead of characters.) Like lines, the actual space can vary depending on your output device, so you would only use it when the character count is the critical element in your measurement.

**ems** A number of ems, where em is the width or the height of the character "m" in the current font.

For the 3800 Model 1, this width may be:

| | |
|---|---|
| 1/10 inch | (2.54 millimeters) |
| 1/12 inch | (2.117 millimeters) |
| 1/15 inch | (1.693 millimeters) |

depending on the "pitch" (the number of characters per inch) of the font being used.

The height may be:

| | |
|---|---|
| 1/6 LPI | (lines per inch) |
| 1/8 LPI | |
| 1/12 LPI | |

Ems can be specified as a decimal number followed by an "M" or "MH" for horizontal em space, or "MV" for vertical em space, like so:

| | |
|---|---|
| 12m | (12 horizontal ems) |
| 20.5mh | (20 1/2 horizontal ems) |
| 14,25mv | (14 1/4 vertical ems) |

Fractional ems are rounded to the nearest device unit. For line devices, ems are translated to the nearest whole number.

**device units** A number of device units, the size of which depends on the device for which SCRIPT/VS is formatting. Device units are specified as a whole number (no decimal points allowed), followed by a "DH" for horizontal device units, or "DV" for vertical device units, like this:

| | |
|---|---|
| 45dh | (45 horizontal device units) |
| 600dv | (600 vertical device units) |

Because SCRIPT/VS can only actually space vertically in lines for line devices, it converts the other measures to the closest it can come in line spaces. Similarly, the ability to do horizontal spacing depends on the device being used, and SCRIPT/VS comes as close as it can.

For the IBM 4250 printer, SCRIPT/VS can space as small as 1/600 inch, both vertically and horizontally. For the IBM 3820 Page Printer and IBM 3800 Printing Subsystem Model 3 and Model 6, the smallest space possible is 1/240 inch. For the 1403, the closest SCRIPT/VS can space is one character space horizontally and one line space vertically. For other devices, the device units fall in between these two amounts.

As an example, if you specify 60DV on the DEPTH attribute of the FIG tag:

- On the 4250, a space of 1/10 inch would be placed
- On the 1403, a space of 10 inches would be placed.

You can see by the wide range of device units available on the various devices that specifying spacing in device units will very probably make your document *device dependent.* That is, the places in your document where you specify spacing in device units will probably only format correctly when it is printed on the device for which you have specified the device units.

## Exercise: Examples and Figures

See if you can create a document that looks like this (call it EXER3); of course, your figures will come out numbered 1 and 2 instead of what we show here:

Here's an example of some BASIC statements

```
10 PRINT USING 55 A, B, C
20 LET J = K + 2
30 IF J = X GO 80
```

that will solve this problem.

An inline, page-wide figure would look like this:

```
A PAGE-WIDE FIGURE
 Since the contents of a figure
 are formatted EXACTLY as entered,
 you can enter blanks
 on the line       (before text)
    and the lines
    will print
       Exactly
       the same
          as the way
          they were entered!
```

**Figure 3. A Page-Wide Figure:** This is the first figure I have entered myself.

This paragraph follows the FIG end tag. Here we have another figure (inline and column-wide):

```
Now let's create another figure. This one is column-wide.
(This exercise will also create a second item for our listing,
in a future exercise involving a List of Illustrations.)
```

**Figure 4. This Is A Column-Wide Figure**

To see the results back at your terminal (in single column rather than the *offset style* we show you here):

**TSO users enter:**[11]

```
script exer3 prof(dsmprof3) cont
```

EXER3 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

[11] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

**CMS users enter:**[11]

```
  script exer3 (prof(dsmprof3) cont
```

EXER3 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[11]

```
  script * prof(dsmprof3) cont
```

The document to be formatted (EXER3—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

The markup is shown in "Markup for Examples and Figures" on page 195.

# Chapter 5. Creating Tables

Like examples and figures, tables are a method of organizing information and presenting it in a format that can improve both the understanding and the usability of the information.

There are three basic elements in a table:

| | |
|---|---|
| **Cell** | A cell is rectangular and is usually separated from other cells with horizontal and vertical rules. |
| **Row** | A row is a horizontal, rectangular collection of one or more cells. The cells that make up a row may have different widths and depths. |
| **Table** | A table is a collection of one or more rows. The rows that make up a table may contain different cell arrangements. |

The following example shows a table made up of two rows, using the same cell arrangement of three cells per row:

| | | |
|---|---|---|
| This is the first cell in the first row of this table. | This is the second cell of the first row of this table. | This is the third cell. |
| This is the first cell in the second row of this table. | This is the second cell of the second row of this table. | This is the third cell in the second row. |

The attributes on the table tags provide flexibility in specifying different characteristics of the table. For example, you can specify the highlight level to be used in a cell or the vertical alignment of the contents of the cell.

We will show you how to add captions and descriptions to your tables.

You can specify a particular row, called a *header*, to appear at the top of every column on every page on which the table appears. Likewise, you can specify a particular row, called a *footer*, to appear at the bottom of every column on every page on which the table appears.

## Defining the Table

Before you can create a table, you need to define its *attributes* using the RDEF tag. These attributes:

- Provide a name for the row definition: ID (this is the only required attribute)
- Specify the number, arrangement, and size of the cells in the row: CWIDTHS and ARRANGE
- Specify the characteristics of the cells in the row: HP, ALIGN, CONCAT, VALIGN, ROTATE, and MINDEPTH.

For this last set of attributes, more than one value can be specified. The first value specified will apply to the first cell, the second value to the second cell, and so on. If fewer values are specified than the number of cells, the remaining cells will use the last value specified. If a greater number of values are specified than the number of cells, the extra values will be ignored. If multiple values are given, the entire string of values must be enclosed in single quotation marks. This technique is shown in examples later in this chapter. The values for all of these attributes (except ROTATE and MINDEPTH)[12] can be abbreviated. For example, on the ALIGN attribute, the values "LEFT", "LEF", "LE", and "L" are all valid ways to specify left alignment for the contents of the cell.

The following shows the attributes used on the RDEF tag:

```
:RDEF ID=name
      HP=highlight level
      ALIGN=horizontal alignment
      CONCAT=YES|NO
      VALIGN=vertical alignment
      ROTATE=rotation
      MINDEPTH=vertical space
      ARRANGE=arrangement of cells
      CWIDTHS=horizontal space
```

where:

**ID** = name

is the attribute that allows you to refer to this row definition when starting a table, row, header, or footer. This identifier is not case-sensitive and can be a mixture of letters and numbers, up to seven characters in length. This attribute is required.

**HP** = highlight level

specifies the highlighting level to be used for the text in the cells of your table. The allowable values are 0, 1, 2, and 3.

The four allowable values correspond directly to the four levels of highlighting available with the HP tag.

This attribute is optional and the default is 0. For more information about highlighting, see "Chapter 7. Highlighting, Citing, Noting, and Quoting" on page 71.

**ALIGN** = horizontal alignment

determines the horizontal alignment of the contents of your cells. This attribute is optional and the default is LEFT. You can specify any of six choices for horizontal alignment:

| | |
|---|---|
| **LEFT** | Aligns the cell contents with the left-hand side of the cell. |
| **CENTER** | Aligns the cell contents in the middle of the cell. |
| **RIGHT** | Aligns the cell contents with the right-hand side of the cell. |
| **INSIDE** | On odd-numbered pages, aligns the cell contents with the left-hand side of the cell. On even-numbered pages, aligns the cell contents with the right-hand side of the cell. |

[12] The ROTATE and MINDEPTH attributes use numbers and vertical space unit notations as values, so these values cannot be abbreviated.

| | |
|---|---|
| **OUTSIDE** | On odd-numbered pages, aligns the cell contents with the right-hand side of the cell. On even-numbered pages, aligns the cell contents with the left-hand side of the cell. |
| **JUSTIFY** | Inserts additional "white space" between words in the cell, producing even left and right edges. |

**CONCAT** = *YES*|NO

determines whether or not the input lines will be concatenated when producing output lines. This attribute is optional and the default is YES.

| | |
|---|---|
| **YES** | Specifies that the input lines will be concatenated to produce output lines which are as full as possible |
| **NO** | Specifies that the printed output lines will match the way they were entered. |

**VALIGN** = vertical alignment

determines the vertical alignment of the contents of the cells. This attribute is optional and the default is TOP. You have three choices for vertical alignment:

| | |
|---|---|
| **TOP** | Indicates the contents of the cell are to be placed at the top of the cell |
| **CENTER** | Indicates the contents of the cell are to be centered vertically in the cell |
| **BOTTOM** | Indicates the contents of the cell are to be placed at the bottom of the cell. |

**ROTATE** = rotation

specifies the rotation of the contents of the cell. The allowable values are 0, 90, -270, 180, -180, 270, and -90. This attribute is optional and the default is 0.

**Note:** This attribute is only valid for the IBM 3800 Printing Subsystem Model 3 and Model 6 and the IBM 3820 Page Printer. A composite rotation of 180 or -180 degrees is not allowed for the IBM 3800 Printing Subsystem Model 3 or Model 6.

If you specify a rotation of 90, -270, 270, or -90 the MINDEPTH attribute is required for that cell. See the description of MINDEPTH for more information.

This same attribute is used on the TABLE tag to specify rotation of the entire table.

**MINDEPTH** = vertical space

specifies the minimum depth of a cell. If the content of a cell does not fill the specified minimum depth, "white space" is automatically added at the top and/or at the bottom of the cell (depending on the VALIGN value for that cell) to make the cell the specified minimum depth. This attribute is optional for cells rotated 0, 180, or -180. If this attribute is not specified for a particular cell, or if an asterisk (*) is specified, the contents of the cell and the depth of other cells in the row will determine the depth of the cell.

If you specify a rotation value of 90, -270, 270, or -90 for a cell, then the MINDEPTH attribute is required for that cell. That's because the MINDEPTH value will be used as the column line length for the contents of the cell. The actual depth of such a cell may be greater than the MINDEPTH value specified for that cell depending on the depth of other cells in the row. However, the contents of the cell will still be formatted using the MINDEPTH value. If this produces undesirable formatting results, increase the MINDEPTH value for the cell.

**ARRANGE** = arrangement

specifies the arrangement of the cells in a row. You must enter positive integers as values on this attribute. The values you enter correspond to the cell numbers used with the C tag. This attribute is optional and the default is *n* cells, where *n* is the number of values given on the CWIDTHS attribute. If no CWIDTHS attribute is specified, the default is one cell.

See "How to Use the ARRANGE and CWIDTHS Attributes" on page 55 for a method of determining the ARRANGE and CWIDTHS values for a cell arrangement.

**CWIDTHS** = horizontal space

is used to determine the widths of the individual cells. Use any of the standard notations for horizontal space units. (See "Space Units" on page 34 for the allowable space unit notations.)

If the ARRANGE attribute is not used, the CWIDTHS attribute determines the number of cells in the row, and the width of each cell. There will be one cell for each value specified on the CWIDTHS attribute. The values specified in the CWIDTHS attribute are used as the widths of the cells.

**Note:** If you use the ARRANGE attribute, the number of values specified on the CWIDTHS attribute must be either:

- The number of values given between slashes, or,
- The number of values on one ARRANGE attribute (If slashes are not used).

The CWIDTHS values indicate the widths of the individual rectangles in the grid that were defined with the ARRANGE attribute. The width of a particular cell will then be the sum of the CWIDTHS values for all the grid rectangles which comprise the particular cell. See "How to Use the ARRANGE and CWIDTHS Attributes" on page 55 for a method of determining the ARRANGE and CWIDTHS values for a cell arrangement.

The asterisk (*) can also be used as an attribute value. This notation indicates that any remaining horizontal space be divided equally among the number of asterisks you enter on that particular attribute. A number can precede the asterisk to indicate a factor to apply. For example, a cell with a width of "2*" will be twice as wide as a cell with a width of "*."

If the CWIDTHS attribute is not used, the default is ***n*** cells of equal width, where ***n*** is the number of values given between one set of slashes on the ARRANGE attribute or on a single ARRANGE attribute if slashes are not used.

## *Building the Table*

Once you have defined the rows in your table, use the TABLE tag to start the table. Use the ROW tag and the C tag to begin the rows and cells of your table. Use the TABLE end tag to end your table.

The TABLE tag has seven attributes:

```
:TABLE REFID=name
       ID=name
       WIDTH=horizontal space
       COLUMN
       PAGE
       SPLIT=NO|YES
       ROTATE=rotation
```

where:

**REFID** = name

refers to a unique row definition previously defined with the RDEF tag. This attribute is required. It is not case-sensitive.

**ID** = name

is the attribute that allows you to make cross-references to this table with the TREF tag. (Cross-referencing is described in "Chapter 6. Cross-References and Footnotes" on page 63.) This identifier can be a mixture of letters and numbers up to seven characters in length. The identifier is case-sensitive.

**COLUMN**

sets the width of the table to the current column line length. The COLUMN and PAGE attributes are mutually exclusive. The last one specified will be used.

**PAGE**

sets the width of the table to the current line length. The COLUMN and PAGE attributes are mutually exclusive. The last one specified will be used. *PAGE* is the default.

**WIDTH** = horizontal space

specifies the width of the table. The value specified can be any of the horizontal space units described in "Space Units" on page 34. If this attribute is not specified, the width of the table will be determined by the COLUMN or PAGE attribute.

**SPLIT** = *NO*|YES

specifies whether or not a table can be split between rows when the table is too large to fit in a single column. You can also specify the SPLIT attribute on the ROW tag. See "Starting a Row" on page 46 for more information.

**NO** Specifies that if the table will not fit in the current column, it cannot be split except prior to rows that have SPLIT = YES specified on the ROW tag. The default is *NO*.

**YES** Specifies that if the table will not fit into the current column, the table can be split prior to any row, except for rows that have specified SPLIT = NO on the ROW tag.

If a table will not fit on the current page, and there are no rows that can be split, the entire table is moved to the next page. If the table will not fit on the next page, a warning message will be issued, and the table will be split, fitting as many rows on the current page as possible.

**ROTATE** = rotation

specifies a rotation for the entire table. The allowable values are 0, 90, -270, 180, -180, 270, and -90. This attribute is optional—the default is 0.

This attribute is valid only for the IBM 3800 Printing Subsystem Model 3 and Model 6 and for the IBM 3820 Page Printer.

A table will not be rotated when the COLUMN attribute has been specified.

**Note:** A composite rotation of 180 or of -180 is not allowed for the IBM 3800 Printing Subsystem Model 3 or Model 6.

## Starting a Row

Start each row of a table with a ROW tag. The cells in each row are formatted using the row definition values you previously entered on the RDEF tag.

A ROW end tag, which is optional, can be used to end a row. Another row, table note, table caption, table description, or a TABLE end tag will also end the current row.

There are two attributes for this tag:

```
:ROW REFID=name
     SPLIT=NO|YES
```

where:

**REFID** = name

refers to a unique row definition previously defined with the RDEF tag. This attribute is optional. It is not case-sensitive. If not specified, the row definition from the previous row (or the TABLE tag if this is the first row in the table) will be used.

**SPLIT** = *NO*|YES

specifies whether or not the table can be split just ahead of this row.

**NO** Specifies that if the table will not fit in the current column, it cannot be split prior to this row. The default is *NO*.

**YES** Specifies that if the table will not fit in the current column, the table can be split prior to this row.

If the SPLIT attribute is not specified, the SPLIT value used on the TABLE tag will determine whether or not the table can be split prior to this row.

If a table will not fit in the current page and there are no rows that can be split, the entire table is moved to the next page. If the table will not fit on the next page, a warning message will be issued and the table will be split, fitting as many rows on the current page as possible.

## Starting a Cell

Use the C tag to start each cell in a row of a table, followed by the text you want placed in that cell. The cell contents can continue for more than one input line. The C end tag, which is optional, can be used to end a cell. A cell can also be ended by another cell tag or by another row, table note, table caption, table description, row end tag, table end tag, table header end tag, or table footer end tag.

This tag has one attribute:

```
:C n
```

where:

**n**

is the *number* of the cell to be started.

This attribute is optional. If a cell number is not specified, the cell with the next highest cell number in the row will be started. If no such cell exists, the current row will be ended and a new row will be started using the same row definition, and the cell with the lowest cell number in that row will be started.

If you attempt to place document elements (text or page segments) outside a cell, the starter set ends the table and issues an error message.

## The Table Header

The THD tag is used to define a header for the table. A table header is a special row placed at the top of the table in all columns or pages on which the table appears. The THD end tag is required to end the table header definition, and it must begin in column one of an input line.

The THD tag has one optional attribute:

```
:THD REFID=name
```

where:

**REFID** = name

refers to a unique row definition previously defined with the RDEF tag. This attribute is optional. If not specified, the row definition from the previous row (or the TABLE tag if this is the first row in the table) will be used. This attribute is not case-sensitive.

The THD tag is only valid within a table. The table header must be defined before you start the first row in the table. Use the C tag to fill the cells in your table header, just like you did to fill the cells in rows.

## The Table Footer

The TFT tag defines a footer for the table that appears at the bottom of the table in all columns or pages on which the table appears. The TFT end tag is required to end the table footer definition, and it must begin in column one of an input line.

The TFT tag has one optional attribute:

```
:TFT REFID=name
```

where:

**REFID** = name

refers to a unique row definition previously defined with the RDEF tag. This attribute is optional. If not specified the row definition from the previous row (or the TABLE tag if this is the first row in the table) will be used. This attribute is not case-sensitive.

The TFT tag is only valid within a table. The table footer must be defined before you start the first row in the table. Use the C tag to fill the cells in your table footer, just like you did to fill the cells in rows.

## The Table Caption and Table Description

If you want a caption for your table, use the TCAP tag at the end of your table, prior to the TABLE end tag. Your table must have a caption in order to appear in the list of tables created with the TLIST tag. The text of the caption must appear on the same line as the TCAP tag—or entirely on the next line—and there must be no GML tags in the caption.

You do not have to enter the word "Table" or the table number. The starter set takes care of that for you.

If you have a lot that you want to say about the table, you can also have a table description, which can go on for as many lines as you need. The table description text will not be placed in the list of tables but will be added to the table caption at the bottom of the table.

Place the TDESC (table description) tag immediately after the TCAP tag and before the TABLE end tag.

The TCAP tag:

- Numbers the table automatically
- Formats the text that was included with the TCAP tag
- Places the table number and text from the TCAP tag into the list of tables.

The TDESC tag:

- Places a colon (:) after the text of the table caption
- Follows the colon with the text of TDESC tag.

**Note:** Because the table caption and description are not actually part of the table, they might be moved to a different column or page if the table itself is large enough to fill the current column.

## Table Notes

Use the TNOTE tag to insert a note in a table. The TNOTE tag will start a new row consisting of only one cell that is as wide as the table. The word: "**Note:**" in a level-two highlighted phrase (HP2) will precede the note text you enter.

Use the TNOTE end tag to end a table note. The tag will also cause the row to be ended, so you will need to specify another ROW tag to start a new row.

## *Sample Tables*

On the following pages we show a number of sample tables illustrating various attributes of the table tags. There are examples of tables with different row definitions, different kinds of text highlighting, text alignments, and cell rotations. We also include tables that illustrate table notes, table captions, and table descriptions.

## Three-Row Table with Text in the Cells

Here is an example of a very simple table consisting of three rows; all using the same row definition. The row definition does not need an ARRANGE attribute because all of the cells in the row extend from the top to the bottom of the row. The COLUMN attribute will be used on the TABLE tag to cause the table to be as wide as the current column, not the current page:

| | | |
|---|---|---|
| Here's the first cell | This text is in cell No. 2 | Cell #3 |
| Here's the first cell in the second row | The arrangement of this row is the same as the previous row and the next row | The depth of the row is determined by the depths of the cells, so this row ends up being deeper then the previous and next rows. |
| This is the third row. | | Cell 2 in this row is empty. |

Here are the tags used to define and create this table:

```
:rdef id=exmpone cwidths='2i 2i 1i'.
:table column refid=exmpone width=5i.
:row.
:c.Here's the first cell
:c.This text is in cell No. 2
:c.Cell #3
:row.
:c.Here's the first cell in the second row
:c.The arrangement of this row is the
same as the previous row and
the next row
:c.The depth of the row is determined by the
depths of the cells, so this row ends up being
deeper then the previous and next rows.
:row.
:c.This is the third row.
:c 3.Cell 2 in this row is empty.
:etable.
```

## Table with Two Different Row Definitions

The following table consists of four rows. The first three rows use one row definition and the fourth row uses a different row definition. We did not specify either the COLUMN or WIDTH attributes, so the current line length will determine the width of the table.

| | | |
|---|---|---|
| Here's the first cell | This text is in cell No. 2 | Cell #3 |
| Here's the first cell in the second row | The arrangement of this row is the same as the previous row and the next row | The depth of the row is determined by the depths of the cells, so this row ends up being deeper than the previous and next rows. |
| This is the third row. | | Cell 2 in this row is empty. |

| | | | |
|---|---|---|---|
| This row has a different arrangement than the pre- vious row. | This is the second cell. | This is the third cell, nothing will be placed in the fourth cell. | |

Here are the tags used to define and create this table:

```
:rdef id=exmpone cwidths='2i 2i *'.
:rdef id=exmptwo cwidths='1i * * *'.
:table refid=exmpone.
:row.
:c.Here's the first cell
:c.This text is in cell No. 2
:c.Cell #3
:row.
:c.Here's the first cell in the second row
:c.The arrangement of this row is the same
as the previous row and the next row
:c.The depth of the row is determined by the depths
of the cells, so this row ends up being deeper
than the previous and next rows.
:row.
:c.This is the third row.
:c 3.Cell 2 in this row is empty.
:row refid=exmptwo.
:c.
This row has a different arrangement than the previous row.
:c.This is the second cell.
:c.This is the third cell, nothing will
be placed in the fourth cell.
:etable.
```

## Changing the Highlight Level of the Cells

The highlight level used for the text in a cell can be specified by using HP attribute on the RDEF tag.

| | | | |
|---|---|---|---|
| This is the first cell, it uses highlight level 0. This is the same as specifying the HP0 tag. | *This cell was defined to be in highlight level 1.* | **This cell was defined to be in highlight level 2.** | **There were only 3 specifications on the HP attribute, so this cell, the fourth cell, will use the highlight level used in the previous cell.** |

Here are the tags used to define and create this table:

```
:rdef id=more cwidths='1i * * *'
      hp='0 1 2'.
:table refid=more column.
:row.
:c.This is the first cell, it uses highlight level 0.
This is the same as specifying the HP0 tag.
:c.This cell was defined to be in highlight level 1.
:c.This cell was defined to be in highlight level 2.
:c.There were only 3 specifications on the HP attribute, so this
cell, the fourth cell, will use the highlight level used in the
previous cell.
:etable.
```

## More Cell Characteristics

Other cell characteristics you can specify on the RDEF tag are: horizontal alignment (ALIGN attribute), vertical alignment (VALIGN attribute), and concatenation (CONCAT attribute). The following table uses all three of these attributes:

| | | | |
|---|---|---|---|
| This cell uses highlight level 0. Concat-enation is on in this cell, all the lines will be left aligned, and the contents of this cell will be aligned to the top of the cell | *Highlight level 1, text lines centered and not concatenated, bottom vertical alignment is used.* | **This cell was defined to be in highlight level 2. Concat-enation of the text lines is done, and right-alignment is done. The cell con-tents appear at the top of the cell.** | **There were 3 values on the HP attribute, so the fourth cell will use the highlight level used in the pre-vious cell. Input lines are glued to-gether and space added between words to make the left and right edges of this text even. The text of this cell is cen-tered vertically.** |

Here are the tags used to define and create this table:

```
:rdef id=more cwidths='1i * * *'
      hp='0 1 2' align='l c r jus'
      valign='top bottom top center' concat='yes no yes'.
:table refid=more column.
:row.
:c.This cell uses highlight level 0.
Concatenation is on in this cell, all the lines will be left
aligned, and the contents of this cell will be aligned to the top
of the cell
:c.Highlight
level 1,
text lines
centered and
not
concatenated,
bottom vertical
alignment is used.
:c.This cell was defined to be in highlight level 2.
Concatenation of the text lines is done,
and right-alignment is done.
The cell contents appear at the top of the cell.
:c.There were 3 values on the HP attribute, so
the fourth cell will use the highlight level used in the
previous cell.
Input lines are glued together and space added between words to
make the left and right edges of this text even.
The text of this cell is centered vertically.
:etable.
```

## Cell Rotation and Minimum Depth

When formatting for the 3800 Print Subsystem Model 3 and Model 6 and the 3820 Page Printer, you can rotate cells within a table by using the ROTATE attribute on the RDEF tag. When specifying a rotation of 90 or 270 degrees, you must also specify a minimum depth with the MINDEPTH attribute. If you specify a cell rotation of 90, -270, 270, or -90, the MINDEPTH attribute determines the width of the formatted text within the cell. Just for fun, the cell number attribute will be used on the C tags in order to fill the cells in a non-sequential order:

| | | | |
|---|---|---|---|
| This cell is rotated 90 degrees. The MINDEPTH value was 3 inches, so the depth of the cell is 3 inches, which in this case is the width of the text in the cell. The text is centered vertically in the cell. | This cell is rotated 0 degrees. The text is centered vertically in the cell. | This cell is rotated 180 degrees. The text is centered vertically in the cell. | This cell is rotated 270 degrees, the width of the text in the cell is 3 inches. The text is centered vertically in the cell. |

Here are the tags used to define and create this table:

```
:rdef id=bbob rotate='90 0 180 270'
      mindepth='3i * * 3i'
      valign=center
      cwidths='* * * *'.
:table column refid=bbob.
:row.
:c.This cell is rotated 90 degrees.
The MINDEPTH value was 3 inches,
so the depth of the cell is 3 inches,
which in this case is the width of the text in the cell.
The text is centered vertically in the cell.
:c 3.This cell is rotated 180 degrees.
The text is centered vertically in the cell.
:c 2.This cell is rotated 0 degrees.
The text is centered vertically in the cell.
:c 4.This cell is rotated 270 degrees, the width of the text in
the cell is 3 inches.
The text is centered vertically in the cell.
:etable.
```

## Creating a Table Note, Caption, and Description

In the following table the TNOTE, TCAP, and TDESC tags were used to create a table note, caption, and description:

| | | |
|---|---|---|
| This is the first cell; the second and third cells will be empty | | |
| **Note:** This is a table note. It will be placed in a row containing one cell that is as wide as the table | | |
| | | This is the third cell in this row. (The first and second cells are empty.) |

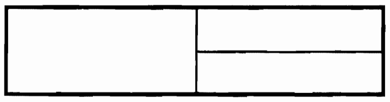
**Table 1. Example of a Table Note, Caption and Description:** This is a rather short example. It is probably not representative of a table you might create for a real purpose.

Here are the tags we used to define and create this table:

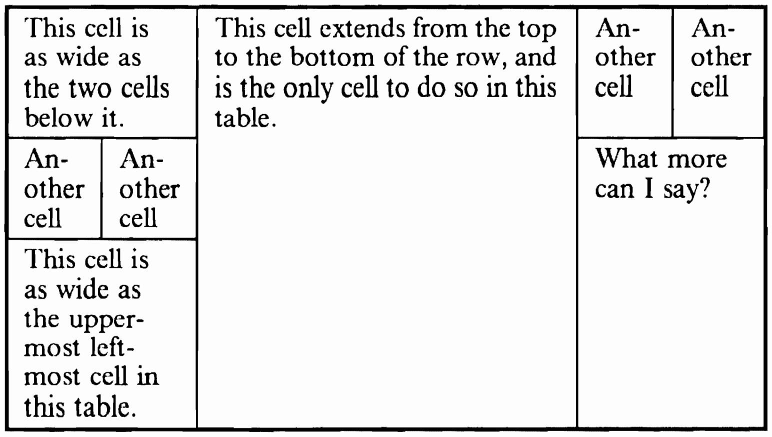
```
:rdef id=geef cwidths='3* 2* *'
:table refid=geef.
:row.
:c.This is the first cell; the second and third
cells will be empty
:tnote.This is a table note. It will be placed in a row
containing one cell that is as wide as the table
:etnote.
:row.
:c 3.This is the third cell in this row. (The first and second
cells are empty.)
:tcap.Example of a Table Note, Caption and Description
:tdesc.This is a rather short example. It is probably
not representative of a table you might create for
a real purpose.
:etable.
```

## How to Use the ARRANGE and CWIDTHS Attributes

In the preceding examples, we didn't use the ARRANGE attribute. If all the cells in a row extend from the top of the row to the bottom of the row, the ARRANGE attribute is not necessary because the CWIDTHS attribute determines the width of each cell. The cells are numbered from left to right, starting with cell number 1. The ARRANGE attribute is necessary when you are defining a row which contains at least one cell that does not extend from the top of the row to the bottom of the row. The following is the simplest example of such a row:



Consider the table:

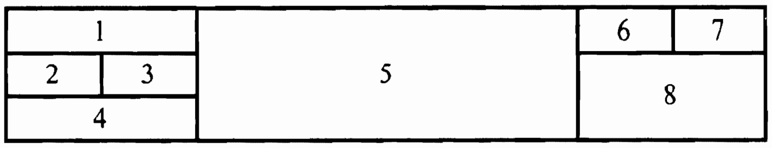


| | | | | |
|---|---|---|---|---|
| This cell is as wide as the two cells below it. | | This cell extends from the top to the bottom of the row, and is the only cell to do so in this table. | Another cell | Another cell |
| Another cell | Another cell | | What more can I say? | |
| This cell is as wide as the uppermost leftmost cell in this table. | | | | |

Determining the correct ARRANGE attribute values to create this table can be tricky. The following steps should make it a little easier.

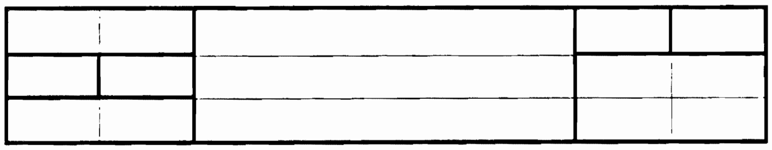
### *Step 1. Sketch the Row*

Sketch your row and number each cell using positive integers. You do not have to number the cells sequentially but the process is easier if you do.



### *Step 2. Form a Grid*

Extend all the vertical and horizontal rules to the edges of the sketch to form a grid.



The extensions are shown in thin rules and the cells from Step 1 are shown in thick rules.

### *Step 3. Determine the Widths of the Cells*

Determine the horizontal width of each rectangle in the grid from Step 2. These are the values to use on the CWIDTHS attribute.

| 3p | 3p | 12p | 3p | 3p |
|---|---|---|---|---|
| | | | | |
| | | | | |

The CWIDTHS attribute for this row would be:

```
CWIDTHS='3p 3p 12p 3p 3p'
```

The CWIDTHS values add up to 24 picas, so the following WIDTH attribute should be specified on the TABLE tag:

```
WIDTH=24p
```

### *Step 4. Number Each Box*

Number each rectangle in the grid with the number of the cell from Step 1. In the first line of the grid, place a '1' in the first and second rectangles (because they make up cell 1 in Step 1). Your grid should look like this:

| 1 | 1 | 5 | 6 | 7 |
|---|---|---|---|---|
| 2 | 3 | 5 | 8 | 8 |
| 4 | 4 | 5 | 8 | 8 |

In this example the numbers from 1 to 8 were used in sequential order. Remember, you can use any integers, in any order, as long as the cell composed of grid elements with the same number forms a rectangle.

### *Step 5. List the Numbers*

List the numbers as they appear in the grid elements from Step 4:

```
1  1  5  6  7
2  3  5  8  8
4  4  5  8  8
```

Then, use these numbers with the ARRANGE attribute:

```
ARRANGE='1 1 5 6 7'
ARRANGE='2 3 5 8 8'
ARRANGE='4 4 5 8 8'
```

or

```
ARRANGE='1 1 5 6 7 / 2 3 5 8 8 / 4 4 5 8 8'
```

### *Step 6. Enter the Row Definition*

The complete row definition looks like this:

```
:rdef id=myrow arrange='1 1 5 6 7'
               arrange='2 3 5 8 8'
               arrange='4 4 5 8 8'
               cwidths='3p 3p 12p 3p 3p'.
```

## Using the ARRANGE Attribute

The following is an example of a table requiring the ARRANGE attribute:

| Selected major industry and selected country | Number of U.S. corporation returns | Controlled Foreign Corporations | | | |
|---|---|---|---|---|---|
| | | Number of foreign corporations | Total assets of corp. | Foreign corporations with current earnings and profits (+) before taxes | |
| | | | | Current earnings and profits before taxes | Foreign income taxes (net) |

Here are the tags used to define and create this table:

```
:rdef id=corptbl cwidths='1i * * * * *'
     arrange='1 2 3 3 3 3'
     arrange='1 2 4 5 6 6'
     arrange='1 2 4 5 7 8'
     align='left left center center center left center'
     concat='yes yes no yes'
     valign='center center top center center top bottom'.
:table refid=corptbl column.
:row.
:c.Selected major industry and selected country
:c.Number of U.S. corporation returns
:c.Controlled Foreign Corporations
:c.Number of foreign corporations
:c.Total assets of corp.
:c.Foreign corporations with current earnings and
profits (+) before taxes
:c.Current
earnings and
profits before taxes
:c.Foreign income taxes (net)
:etable.
```

## Using a Table Header

A table header is an effective way of defining a row that describes the contents of the cells in the rows below it. The following table also uses the TCAP and TDESC tags to create a table caption and description at the end of the table:

| **Type of Offense** | **Number of Offenders** | **Number of Offenses** | **Distance from Offender's Residence** | | | | |
|---|---|---|---|---|---|---|---|
| | | | **Less than 1 Block** | **1 to 2 Blocks** | **2 to 3 Blocks** | **3 to 4 Blocks** | **Unknown** |
| **Trespassing** | 02 | 03 | 40 | 20 | | | |
| **Carrying Weapons** | 05 | 04 | 29 | 30 | | | |
| **Drinking, disorderly conduct, glue-sniffing, creating disturbance** | 07 | | 10 | | 40 | | 30 |
| **Subtotal** | 14 | 07 | 79 | 50 | 40 | | 30 |

Table 2. **Place of Offense and Residence of Offender:** This table indicates the correlation between different types of criminal offenses, and the distance from the offender's residence to the scene of the offense.

Here are the tags used to define and create this table:

```
:rdef id=hdr cwidths='1.75i' concat=n hp=2
      arrange='1 2 3 4 4 4 4 4 / 1 2 3 5 6 7 8 9'
      align='center left left center left' valign=center
      rotate='0 270 270 0 270' mindepth='* 2i 2i * 1.5i'.
:rdef id=body cwidths='1.75i * * * * * * *'
      hp='2 0' align='left center'.
:table refid=body column.
:thd refid=hdr.
:c.Type of Offense
:c.Number of Offenders
:c.Number of Offenses
:c.Distance from Offender's Residence
:c.Less than 1 Block
:c.1 to 2 Blocks
:c.2 to 3 Blocks
:c.3 to 4 Blocks
:c.Unknown
:ethd.
:row refid=body.
:c.Trespassing
:c.02:c.03:c.40:c.20
:row.
:c.Carrying Weapons
:c.05:c.04:c.29:c.30
:row.
:c.Drinking, disorderly conduct, glue-sniffing,
creating disturbance
:c.07:c 4.10:c 6.40:c 8.30
:row.
:c.Subtotal
:c.14:c.07:c.79:c.50:c.40:c 8.30
:tcap.Place of Offense and Residence of Offender
:tdesc.This table indicates the correlation between different
types of criminal offenses, and the distance from the
offender's residence to the scene of the offense.
:etable.
```

## *Exercises: Tables*

On this page and the next are two exercises for you to practice using the starter set table tags.

### *First Exercise*

See if you can create a table that looks like the one below (call the file EXERT1):

| Year | Passenger Cars | Trucks and Buses | *Total* |
|---|---|---|---|
| 1900 | 4,192 | | 4,192 |
| 1905 | 24,250 | 750 | 25,000 |
| 1910 | 181,000 | 6,000 | 187,000 |
| 1915 | 895,930 | 74000 | 969,930 |
| 1920 | 1,905,560 | 321,789 | 2,227,349 |

**Table 3.** **U.S. Automobile Production:** The above table shows the number of automobiles produced in the United States from the years 1900 through 1920.

To see the results back at your terminal (in single column rather than the *offset style* we show you here):

**TSO users enter:**[13]

```
script exert1 prof(dsmprof3) cont
```

EXERT1 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**CMS users enter:**[13]

```
script exert1 (prof(dsmprof3) cont
```

EXERT1 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[13]

```
script * prof(dsmprof3) cont
```

The document to be formatted (EXERT1—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

The markup is shown in "First Table Exercise" on page 196.

[13] Check with your supervisor or your document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

## *Second Exercise*

Let's try another Exercise, this time called EXERT2. Then check your markup against ours (see answers in "Second Table Exercise" on page 196).

| | |
|---|--:|
| Flag | Delaware |
| | Connecticut |
| | Maine |
| | Maryland |
| | Massachusetts |
| | New Hampshire |
| | New Jersey |
| | New York |
| | North Carolina |
| | Pennsylvania |
| | Rhode Island |
| | Vermont |
| | Virginia |

To see the results back at your terminal (in single column rather than the *offset style* we show you here):

**TSO users enter:**[14]

```
script exert2 prof(dsmprof3) cont
```

EXERT2 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**CMS users enter:**[14]

```
script exert2 (prof(dsmprof3) cont
```

EXERT2 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[14]

```
script * prof(dsmprof3) cont
```

The document to be formatted (EXERT2—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

Check your markup against ours (see answers in "Second Table Exercise" on page 196).

[14] Check with your supervisor or your document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

# Chapter 6. Cross-References and Footnotes

The ID attribute assigns a name to sections of information (or document elements) so that the identified information can be referred to in another part of the document.

## *To Headings*

You've already seen cases in this book of cross-references to headings, although you may not have realized it at the time. For example, this is a cross-reference:

See "Chapter 3. Creating Lists" on page 15.

To get the cross-reference, two things had to be done. First of all, the heading that we referred to was entered like this:

```
:h1 id=clst.Creating Lists
```

The "clst" is a name we made up; it could be any combination of letters and numbers as long as it is no more than 7 characters and the first character is a letter; it is a good idea to always use lowercase letters. (Of course, you couldn't have another heading with the same name, because the starter set wouldn't know which one you were talking about.)

The other thing that had to be done was to enter the cross-reference itself using the HDREF (heading reference) tag with a REFID attribute. It looks like this:

```
:p.See :hdref refid=clst..
```

Notice the two periods at the end of the example; the first one ends the markup and the second one is a normal period that ends our sentence. A common error is to forget that you need two periods at that point.

Because we used the same value for the REFID attribute as for the ID attribute of the heading we are referring to, the starter set knows which one we want. Because it also knows what page it is on, it supplies that, too. If the heading had been on the same page as the reference, the starter set would know that also, and it would not give the page number.

You can use HDREFs to refer to any level heading as long as the heading you refer to is given an ID attribute when it's entered.

It is a good idea to pick descriptive names like *summary* and *intro* for your ID attributes. If you name things *chap1*, *chap2*, and so on, you will find that when you update the source file and insert, delete, and replace material, your names will be more confusing than useful in trying to keep track of what is going on.

There are times when you want to leave out the page number even though the heading is on a different page. The starter set lets you do this with the PAGE attribute on the HDREF tag. Your markup would look like this:

```
:p.See :hdref page=no refid=clst..
```

On the other hand, if you wanted the page reference included even if the reference lands on the same page as the heading, you could enter the PAGE attribute with a value of "yes".

Cross-references can go in either direction; that is, you can refer to information identified with the ID attribute that comes before or after the cross-reference. However, if the information being referred to comes after the reference, you must specify two or more formatting passes with the FPASSES or TWOPASS option of the SCRIPT command at run time for the reference to appear. If you don't, the starter set will fill in "-- Heading id 'nohead' unknown --" where you wanted the cross-reference. We'll tell you more about FPASSES and TWOPASS in "Chapter 12. Things You Can Specify At Run Time" on page 115.

## *To List Items*

List item references are the same as heading, figure, and table references except you use the LIREF (list item reference) tag.

We'll create a short, ordered list here to demonstrate how the LIREF tag works.

---

Steps to follow when changing wallpaper:

1. Peel off the old wallpaper.
2. Clean and size the wall.
3. Prepare the paste.
4. Get help from someone who knows what he's doing.

---

We've chosen an ordered list item to cross-reference because that is the one you'd be most likely to refer to. You can refer to a list item from any of the lists, but because simple lists and unordered lists have no numbers, the reference wouldn't make much sense.

Here is the markup for our example:

```
Steps to follow when changing wallpaper:
:ol.
:li.Peel off the old wallpaper.
:li.Clean and size the wall.
:li.Prepare the paste.
:li id=help.Get help from someone
who knows what he's doing.
:eol.
```

Now we can say:

```
:p.We'd better start with number
:liref refid=help. the next time.
```

and it would appear in our output like this:

We'd better start with number 4. on page 64 the next time.

LIREF allows the PAGE attribute, just like HDREF.

## To Figures

Figure references are just the same as head references, except that you use the FIGREF (figure reference) tag.

Since we don't already have a figure in this book with an ID attribute, let's create one now:

We're partial to figures in
boxes, ourselves.
A figure that is going to
have a cross-reference
*must* also have a caption (FIGCAP).

**Figure 5. Captioned Figure for Cross-Reference**

The caption is needed because the figure number assigned through the FIGCAP tag is what is going to show up in the figure reference (the text of the caption is not used). If you don't have a FIGCAP tag, there is nothing for the cross-reference to refer to.

The only markup we are concerned with from our figure above is the FIG tag itself—you already know how to do the rest of it. The FIG tag looks like this:

```
:fig id=xreffig frame=box place=inline width=column.
```

and there you see the ID attribute.

Now we can say

```
:p.See :figref refid=xreffig. for an illustration of...
```

and it would come out like this:

See Figure 5 for an illustration of...

Because our figure is on the same page as the figure reference, the starter set did not include a page number. However, if they had been on *different* pages, it *would have* given us a page number.

FIGREF also allows the PAGE attribute, just like HDREF and LIREF.

## To Tables

Table references are handled just like heading and figure references, except that you use the TREF (table reference) tag.

Now let's create a table with an ID attribute:

| | | Controlled Foreign Corporations | | | |
|---|---|---|---|---|---|
| | | | | Foreign corporations with current earnings and profits ( + ) before taxes | |
| Selected major industry and selected country | Number of U.S. corporation returns | Number of foreign corporations | Total assets of corp. | Current earnings and profits before taxes | Foreign income taxes (net) |

**Table 4. A Captioned Table, to Demonstrate Cross-Referencing**

The caption is needed because the table number assigned through the TCAP tag is what is going to show up in the table reference (the text of the caption is not used). If you don't have a TCAP tag, there is nothing for the cross-reference to refer to.

The only markup we are concerned with from our table above is the TABLE tag itself–you already know how to do the rest of it. The TABLE tag looks like this:

```
:table refid=corptbl id=tbtest.
```

and there you see the ID attribute.

We could now enter:

```
:p.See :tref refid=tbtest. for an illustration of...
```

and it would come out like this:

See Table 4 for an illustration of...

Because our table is on the same page as the table reference, the starter set did not include a page number. However, if they had been on *different* pages, it *would have* given us a page number.

TREF also allows the PAGE attribute, just like HDREF, LIREF, and FIGREF.

## To Footnotes

A footnote is a little different from the other document elements that you can cross-reference. You can have a cross-reference (callout) to a footnote without using the ID attribute.

In fact, that is probably how you would mark up a footnote most often. When you don't use the ID attribute, the footnote gets referenced where it is defined.

For example:

A footnote reference will appear at the end of this sentence because that is where I'm going to put my footnote.[15]

This is how it was entered:

```
:p.A footnote reference will appear at the end of this
sentence because that is where I'm going to put my footnote.
:fn.
This is the text of a footnote that I entered without using the
ID attribute to give the footnote a name.
:efn.
```

Footnotes can contain paragraphs, lists, highlighted phrases, and so forth. They *cannot* contain examples, figures, or other footnotes.

The footnote is placed at the bottom of the page on which it is entered if there is room for it. If there is room for only one line of the footnote on the current page, all of the footnote is placed on the next page. If at least two lines fit on the current page, the footnote will be split between the two pages.

You'll need to give your footnote a name by using the ID attribute on the footnote tag if you want to make a cross-reference to a footnote from within an example or a figure. The starter set will issue an error message if you put the footnote itself inside a figure or example. The other place you would use the ID attribute is when you want to make more than one callout to a footnote.

Here's how you create a footnote using the ID attribute of the FN tag:

```
:fn id=fnxmp.
This is the text of the footnote.
It starts with an implied paragraph.
:p.It could have several paragraphs, but it
can't contain examples, figures, or another
footnote.
:efn.
```

Check the bottom of the page to see what the starter set has done with our footnote.

And here is our reference.[16]

Of course, you've already figured out what the FNREF tag looks like, but here it is, just in case:

```
:p.And here is our reference.:fnref refid=fnxmp.
```

[15] This is the text of a footnote that I entered without using the ID attribute to give the footnote a name.

[16] This is the text of the footnote. It starts with an implied paragraph.

It could have several paragraphs, but it can't contain examples, figures, or another footnote.

When you need to use the FNREF tag because the footnote reference callout is inside an example or figure, place your footnote just before that example or figure.

## Unmatched References

When the starter set cannot find an ID value that matches the REFID value, it will let you know—where you entered the REFID—that there is no match. Here we've deliberately entered a HDREF, a FIGREF, a LIREF, a FNREF, and a TREF that have unmatched REFIDs ("nohead", "nofig", "nolist", "nofoot", and "notable", respectively) so you can see the output that results.

This markup and text:

```
See :hdref refid=nohead., :figref refid=nofig.,
:liref refid=nolist., :fnref refid=nofoot., and
this nonexistant table :tref refid=notable..
```

produces the following output:

See -- Heading id 'nohead' unknown --, -- LI 'nolist' --,$^{00}$ , -- Figure id 'nofig' unknown --, and this nonexistent table -- Table id 'notable' unknown --.

These messages will also appear when the REFID precedes the ID, and the FPASSES or TWOPASS option is *not* specified on your SCRIPT command. The missing footnote ID is shown with only the zeros to approximate the same length as the footnote reference when it does exist.

The starter set will also include an indication in the cross-reference listing that the reference was not matched (see "Chapter 15. The Cross-Reference Listing" on page 133).

## Listing of Cross-References

The starter set produces a very useful cross-reference listing of all the IDs used in a document. It is placed at the end of the document. You'll find an example of it at the end of this book. (Normally, you'd throw it away before you printed a book, but we've left it for you to look at.)

"Chapter 15. The Cross-Reference Listing" on page 133 tells you what the cross-reference listing contains.

## Exercise: Cross-References and Footnotes

See if you can create a document that looks like the following. Call this one EXER4. Because we promised you we would make our examples "real", you'll have to look at the bottom of the page for the footnote instead of inside the lines around the exercise.

The starter set will also give you a cross-reference listing for this exercise (see "Chapter 15. The Cross-Reference Listing" on page 133). Actually, it gives you one every time (unless you ask it not to) but in our previous exercises, there was nothing to put in it.

### *Heading Being Referred To*

This example will show cross-references to:

1. A figure
2. A footnote
3. A list item
4. A table
5. A heading.

Here's a cross-reference to a figure that comes later, so we need to run with the FPASSES or the TWOPASS option to see the correct results: see Figure 6.

Here we have a footnote reference.[17]

This figure needs referring to.

**Figure 6. Figure Being Referred To**

Don't forget to put an ID attribute on number 3. so that this reference to it resolves properly.

We also want to refer to a table. So, see Table 5.

| Here's | the | table |
|---|---|---|
| being | referred | to |

**Table 5. This part of the exercise is a table**

And finally, we have a heading reference; see "Heading Being Referred To".

(Before we finish this exercise, we must also check to see that all our punctuation is correct.)

[17] Of course, when you run the exercise, the footnote will be numbered 1, since it is the first one in the document.

To see the results back at your terminal (in single column rather than the *offset style* we show you here):

**TSO users enter:**[18]

```
  script exer4 prof(dsmprof3) co twopass
```

EXER4 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**CMS users enter:**[18]

```
  script exer4 (prof(dsmprof3) co twopass
```

EXER4 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[18]

```
  script * prof(dsmprof3) co twopass
```

The document to be formatted (EXER4—referred to by the * in the command) is in your ATMS working storage. DSMPROF3 is in the SYSOP (system operator) permanent storage.

You'll find the markup in "Markup for Cross-References and Footnotes" on page 197.

[18] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization. (Notice that there are three references to this footnote but it only appears once. This is another example of when to use the footnote id attribute rather than let the footnote reference be placed automatically.)

# Chapter 7. Highlighting, Citing, Noting, and Quoting

## *Highlighted Phrases*

You've already seen many examples in this book of text that is *highlighted*. Highlighting—in documents produced on a page printer—is text that is printed in an *italic font*, in a **bold font**, or in a combination of ***italic and bold*** versions of the current font. If you request document printing on a 3800 Model 1, the highlighting is done using underscoring, a bold font, and both underscoring and bold font combined. (The markup, however, is the same no matter which device prints your document.)

We number the highlighting types: 0; *1*; **2**; and ***3***. Highlight 0 is just what you're looking at now—no emphasis.

We have four tags for highlighting, but they are very easy tags to remember, because they all consist of HP (to start the highlighted phrase) plus the number for the type of highlighting you want; that is, to start highlight 2 going, you use HP2 and to end it you use the matching end tag EHP2.

So if you wanted to do something like this:

**Attention!** This is *very* important! ***Don't go out in the rain without your galoshes!***

you would enter it like this:

```
:p.:hp2.Attention!:ehp2.
This is :hp1.very:ehp1. important!
:hp3.Don't go out in the
rain without your galoshes!:ehp3.
```

Remember the HEADHI and TERMHI attributes on definition and glossary lists that we told you about in "Creating Definition Lists" on page 19? The number you specify after the equal sign on the HEADHI and TERMHI attributes (0, 1, 2, or 3) actually refers to the same levels of highlighting that you get with the HP tag. (Of course, if you don't specify these attributes at all, you will get the default values for the highlighting.)

You can also *nest* highlighted phrases—that is, put one kind inside another. The starter set keeps track of where you are, and when you end the inside one, it knows to go back to the one you had going before.

Nesting makes your markup a little easier when you are doing complicated things with highlighted phrases, as we sometimes have to do with programming syntax examples.

For example, to get this result:

**COMMAND KEYWORD = '*DEFAULT*|*value*' KEYWORD**

you would enter:

```
:p.:hp2.COMMAND KEYWORD=':hp3.DEFAULT:ehp3.|:hp1.value:ehp1.'
KEYWORD:ehp2.
```

As you can see from the example, nesting highlighted phrases is a lot less work than it would be to have to keep turning the various highlight types on and off.

Nesting is the reason we have highlight 0 (HP0). Sometimes in a complicated structure, you might want a phrase with no emphasis. Highlight 0 allows you to do that without having to end all the highlights you have going and then start them up again.

## Highlighting for Various Devices

The following paragraphs describe the highlighting available on the various devices SCRIPT/VS produces output for.

Because the 1403 printer (and other printers that work like a 1403) and typewriter terminals do not have a bold font available, highlight 2 and 3 make the type stand out by overstriking the letters several times (highlight 3 also underscores the phrase).

On display terminals, where neither overstriking nor a bold font is available, the starter set shows highlight 2 and 3 by forcing the text to uppercase (highlight 3 is also underscored).

For page devices where many fonts are available, highlighting is done with a change of fonts: italic, bold, and bold italic are used for highlight levels 1, 2, and 3, respectively.

## *Title Citations*

The tags for title citations (CIT and its end tag) are very similar to the highlighting tags.

Title citations are used when you are referring to the title of a book; for example:

Have you read *Gone With the Wind*?

This would be marked up like so:

```
:p.Have you read
:cit.Gone With the Wind:ecit.?
```

Since CIT and HP1 appear to *do* the same thing, you may wonder why we bother to have these extra tags. Perhaps this is a good place to consider again what GML is all about.

By marking things up to describe precisely what they are, you are investing in the future of your text data base.

For example, at some future point you might want to search your text data base for all title citations (for instance, to make sure you aren't citing obsolete titles). Using descriptive tags allows you to do this; the more accurately you describe things, the more useful your data base becomes.

This is why you should not mark something up as, say, an example unless it really is an example, even though today it happens to come out looking like an example. Much of the text being worked on today was created before DSMPROF3 and much more will be created that will still be around long after DSMPROF3 has been retired and newer profiles have taken its place. If you "cheat" in your markup now, you may have to rework the data base later, and none of us wants to do that.

## *Notes*

The starter set has a tag for when you have a note, called NOTE.

A note would be marked up like so:

```
:note.The text of a note is an implied paragraph.
Notes can be used anywhere that paragraphs can be used.
```

It would come out looking like this:

**Note:** The text of a note is an implied paragraph. Notes can be used anywhere that paragraphs can be used.

A note is implicitly ended by a paragraph or a higher-level element.

## *Quotations—the Long and Short of It*

The starter set provides for two different kinds of quotations—inline quotations and excerpts (which we call *long quotations*, although it really has nothing to do with the length).

### Inline Quotations

An inline quotation looks like this:

FDR said, "The only thing we have to fear is fear itself."

The markup for that (using the Q (quoted phrase) tag and its end tag) looks like this:

```
:p.FDR said, :q.The only thing we have
to fear is fear itself.:eq.
```

A first level quote takes double quotation marks; a nested quote within it takes single quotation marks; a nested quote within that takes doubles again, and so forth.

You don't have to decide whether a quote should take single or double quotation marks—DSMPROF3 knows!

You may well ask, "Why use quote tags at all?"

We now have a variety of output devices on which to print our documents. Some of these devices print using fonts that distinguish among open quotation marks, close quotation marks, and apostrophes. By using the proper markup, our files can be printed on any of the available devices with no change to the markup.

## Excerpts (Long Quotations)

An excerpt or long quotation looks like this:

Please distribute the following shipping guidelines to all people in your area who are responsible for handling outgoing packages:

> Packages weighing more than 20 pounds must arrive in the mailroom before 2:30 PM if they are to be shipped the same day; if they are to be shipped air freight, they require an authorization signed by a manager.
>
> Packages that are to be shipped overseas must be submitted to the mailroom with the proper customs clearance forms filled out.

Failure to comply with these guidelines could result in unacceptable delays in mailing out packages.

Notice that the starter set has indented the excerpt on both the right and the left, according to traditional publishing style for handling information quoted from another source.

Here's what the markup (using LQ and its end tag) looks like:

```
:p. Please distribute the following shipping guidelines to all
people in your area who are responsible for handling outgoing
packages:
:lq.
:p. Packages weighing more than 20 pounds must
arrive in the mailroom before 2:30 PM if
they are to be shipped the same day;
if they are to be shipped air freight, they
require an authorization signed by a
manager.
:p.Packages that are to be shipped overseas
must be submitted to the mailroom with
the proper customs clearance forms filled out.
:elq.
:p.Failure to comply with these guidelines could result in
unacceptable delays in mailing out packages.
```

Notice that the LQ tag *does not* imply a paragraph. You must also use a P tag if the first thing in your excerpt is a paragraph. Of course, it could be something else, such as a list.

## *Exercise: Highlighting, Citing, Noting, and Quoting*

Now try creating a document that looks like the following. Call this one EXER5. You'll find the markup in "Markup for Highlighting, Citing, Noting, and Quoting" on page 198.

---

Speak **boldly** on the 3800, **strikingly** on the 1403.

Title references are *also* used for movie titles, such as *Casablanca.* Humphrey Bogart is often misquoted as having said "Play it again, Sam" in that movie. I wish people would learn to quote ***accurately***.

If I said, "he said 'yes' to me," that would be an example of a nested quote.

Look at this excerpt from the *GML User's Guide*:

> You may well ask, "Why use quote tags at all?"
>
> We now have a variety of output devices on which to print our documents. Some of these devices print using fonts that distinguish among open quotation marks, close quotation marks, and apostrophes. By using the proper markup, our files can be printed on any of the available devices with no change to the markup.

**Note:** Notes lose their impact if used excessively.

---

To see the results back at your terminal (in single column rather than the *offset style* we show you here)

**TSO users enter:**[19]

```
script exer5 prof(dsmprof3) cont
```

EXER5 and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**CMS users enter:**[19]

```
script exer5 (prof(dsmprof3) cont
```

EXER5 and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[19]

```
script * prof(dsmprof3) cont
```

The document to be formatted (EXER5—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

---

[19] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

# Chapter 8. Overall Document Structure

So far we've been telling you how to deal with the elements in the body of your document. Now we'll show you how to do things like title pages, tables of contents, and lists of illustrations. Perhaps you also need a security classification on every page of your document.

These tags have more to do with the overall structure of the document than the elements of text within it. The basic document structure tags have no text associated with the tag; rather, they identify a part of the structure of the document.

Let's look for a minute at the overall structure of a general document; it is made up of these pieces:

| | |
|---|---|
| **front matter** | The front matter contains the title page, abstract, preface, table of contents, and list of illustrations. |
| **body** | The body of the document is the main portion of the document. |
| **appendixes** | The appendix section follows the body and contains information supplemental to the material in the body of the document. |
| **back matter** | The back matter contains the glossary and the index. |

The *very first* thing you put in a general document is the GDOC tag. It identifies the document type as *general document*.

The very first line in the source document for this book contains this tag:

```
  :gdoc.
```

If you want a security classification placed at the top of every page, use the SEC attribute, specifying as its value the exact text that you want to appear as the security classification. The text will be centered in level 2 highlighting or a special font for page devices. The security classification will also appear on the title page following the address. The markup might look like this:

```
  :gdoc sec='ABC Company Confidential'.
```

While we're on the subject, we might as well tell you about the *very last* tag you put in a general document. Since you already know the starter set rules for end tags, it must be :EGDOC, right?

So now that we've looked at the beginning and the end of our document, let's look at each of the major pieces in turn.

## The Front Matter

The front matter, specified by the FRONTM tag, contains any or all of these elements, as needed:

- Title page (TITLEP tag)
- Abstract (ABSTRACT tag)
- Preface (PREFACE tag)
- Table of contents (TOC tag)
- List of illustrations (FIGLIST tag)
- List of tables (TLIST tag).

We'll talk about each of these below.

The starter set is very smart about the FRONTM tag: it numbers your front matter pages with lowercase roman numerals, and it knows enough not to put any headings that occur in the front matter into the table of contents.

### The Title Page

The title page follows immediately after the front matter tag. To make up a title page, you enter the TITLEP tag followed by the elements you want on your title page, followed by the TITLEP end tag.

The markup for the sample title page of this document (which you will find behind the IBM cover page up in the front of this book) looks like this:

```
:titlep.
:title stitle='GML Starter Set User''s Guide'
:title.Document Composition...Guide (Sample Title Page)
:docnum.SH20-9186-04
:date.February 26th, 1987
:author.Cliff Hanger
:address.
:aline.Silverado, Colorado
:eaddress.
:etitlep.
```

Each element of the title page is discussed in detail below. All of the elements except for the title page (TITLEP and its matching end tag) are optional. Each element on the title page appears in the following order, regardless of how you enter them:

1. Title
2. Number
3. Date
4. Author
5. Address.

#### *Title*

The document title is entered with the TITLE tag, like so:

```
:title.Document Title
```

The text of the title can't have any tags within it.

The title is printed on the title page and also is used as a running footing on even-numbered pages throughout the document when the document is being formatted for printing in duplex mode (something you specify when you request formatting).

If your title is very long, you will also want to use the STITLE attribute on the TITLE tag, just as we showed for H0 and H1, to get a short title for the even-page running foot. You can see an example of this in our example of the entire title page above.

### *Number*

The document number, if any, is entered with the DOCNUM tag. The text must be either all on the same line as the tag or all on the next line. The text of the DOCNUM tag can't have any tags within it.

### *Date*

The DATE tag causes the starter set to do one of two things:

1. If you enter the tag with no date, the system-supplied date of processing will be placed on the title page each time you format the document.
2. If you enter the DATE tag followed by a date, that date will be printed on the title page.

While this book was being developed, the date tag was used by itself, like so:

```
:date.
```

This way, we could always tell whether a draft was the latest run or not by checking the date on the title page.

When we were ready to publish (and thus would want the same date on all copies, regardless of when they were formatted), we put a date following the date tag, like so:

```
:date.February 26th, 1987
```

The text must be either all on the same line as the tag or all on the next line. The text with the DATE tag can't have any tags within it.

### *Author*

The AUTHOR tag is where you put the author's name. The text must be either all on the same line as the tag or all on the next line and can't have any tags within it.

If you have more than one author, use an AUTHOR tag for each one. All of the authors' names will print on the title page in the order they were entered.

### *Address*

The address of the author or publisher is entered using as many ALINE (address line) tags as needed, surrounded by ADDRESS and its matching end tag. The text for each address line must be either all on the same line as the tag or all on the next line and can't have any tags within it.

Addresses can also be used elsewhere in the document. For example, if you needed to say this:

---

Address technical comments concerning this specification to:

> Technical Specifications Department
> Leading Edge Laboratory
> 2000 State of the Art Boulevard
> Sans-Serif-by-the-Sea, California

---

you would enter it like this:

```
:p.Address technical comments concerning
this specification to:
:address.
:aline.Technical Specifications Department
:aline.Leading Edge Laboratory
:aline.2000 State of the Art Boulevard
:aline.Sans-Serif-by-the-Sea, California
:eaddress.
```

## The Abstract

If your document has an abstract, enter it with the ABSTRACT tag. The ABSTRACT tag has the same effect as a level 1 heading: it begins a new page (an odd-numbered page if you specify duplexing when you request formatting) and generates the heading "Abstract."

An abstract, thus, would be entered like this:

```
:abstract.
:p.This report....
```

## The Preface

If your document has a preface, enter it with the PREFACE tag. The PREFACE tag has the same effect as a level 1 heading: it begins on a new page (an odd-numbered page if you request duplexing when you request formatting) and generates the heading "Preface".

The preface in this book was entered like so:

```
:preface.
:p.The IBM Document Composition Facility....
```

## The Table of Contents, List of Illustrations, and List of Tables

The table of contents, list of illustrations, and list of tables are produced for you by the starter set. All you have to do is put in the TOC, the FIGLIST, and the TLIST tags at those points where you want the table of contents, the list of illustrations and the list of tables to appear.

Thus, the front matter of this book was entered like this:

```
.
.
.
:frontm.
:titlep.
.
.
.
:preface.
.
.
.
:toc.
:figlist.
:tlist.
.
.
.
```

One other thing you need to know about the table of contents, list of illustrations, and list of tables: SCRIPT/VS collects the information that goes into them on every pass through your document. It then puts the final results in your document on the last pass; where you have your TOC, FIGLIST, and TLIST tags. If you run with only one pass, the table of contents, the list of illustrations, and the list of tables will be empty—you'll get the headings for them but nothing else. For more information on the FPASSES or the TWOPASS option, see "FPASSES(n)" on page 121 and "TWOPASS" on page 124.

For review drafts, you could put the TOC, FIGLIST, and TLIST tags at the end of your document (just before the GDOC end tag) so they would be printed on the first pass and you could avoid the need to process the document twice. However, if you do put them in the back matter, they will be in two column format.

**Note:** For information on what to do if your page numbers in the table of contents, the list of illustrations, and the list of tables don't match the actual pages on which the headings and figures or tables appear, see "Chapter 13. Common Problems—Symptoms and Solutions" on page 127.

## *The Body*

When you finish the front matter, you need to identify the body of your document. You do this with the BODY tag. The starter set begins the body of your document, resets the page number to 1 in arabic numerals and puts the headings in the table of contents.

**Note:** We've discussed the basic document elements that you would use in the body of a document earlier in this book.

Here's how it looks for this book:

```
  .
  .
  .
  :toc.
  :figlist.
  :tlist.
  :body.
  :h0 id='intro'.Introduction
  .
  .
  .
```

## The Appendixes

If you have appendixes in your document, use the APPENDIX tag after the body of your document to start the appendix section.

Each H1 tag following the APPENDIX tag is considered to be the start of a new appendix, so the starter set generates the words "Appendix A" for the first one, "Appendix B" for the second, and so forth.

The markup for this book looks like this:

```
  :appendix.
  :h1 id=prim.Creating and Editing a File
  .
  .
  .
  :h1 id=sample.A Sample GML Document
  .
  .
  .
```

You enter the heading in the appendix just as you would in the body; the starter set takes care of making it an appendix for you.

Actually, in doing this book, we also used the tag:

```
  :h0.Appendixes
```

just before our APPENDIX tag. We did this because we wanted a part divider with the heading "Appendixes" on it before our appendixes.

## *The Back Matter*

The back matter of a document typically contains the glossary and index.

Use the BACKM tag to specify that you've gotten to the back matter. The starter set turns off the generation of "Appendix" on your level 1 headings (if you had them going) and sets the back matter into two columns.

In a general document, the index is assumed to be the last thing in the document. Running headings and footings will be suppressed on pages following the index.

In this book, the back matter is entered like so:

```
:backm.
:h1.Glossary
.
.
.
:index.
:egdoc.
```

We'll tell you about the index in "Chapter 9. Indexing" on page 89.

## Summarizing Document Structure

This is how you put all the major elements we've talked about into an overall structure:

```
:gdoc sec='security classification'.
:frontm.
:titlep.
:title stitle='short title'.
Text of Title
:docnum.document number, if needed
:date.either nothing or a specific date
:author.Writer's Name
:address.
:aline.First Line of Address
:aline.And As Many More As You Need
:eaddress.
:etitlep.
:abstract.
.
.
.
:preface.
.
.
.
:toc.
:figlist.
:tlist.
:body
.
.
.
:appendix.
.
.
.
:backm.
.
.
.
:index.
:egdoc.
```

You can use this as a model for creating your own document structures. Anything you don't need you just leave out.

## Combining Input Files

Before we leave the topic of overall document structure, there is one other thing you'll want to know about. It's not really part of the GML markup, but important to know about nonetheless.

SCRIPT/VS allows you to create a single output document from more than one source file.[20] (For example, EXER1 is a source file.) This book was produced using approximately 70 separate source files.

You will want to combine source files, especially for large documents, for several reasons:

1. Different people can work on different parts of the document at the same time if the parts are in separate files.

2. Shorter files are generally more efficient to edit.

3. You can format pieces of the document separately, saving all the overhead of processing the whole document when you only need to check out a piece of it.

4. The same file can be used in any number of output documents. For example, standard text that you put in all documents of a certain type can exist in one source file. If you change the standard, you only need change it in one place.

5. The same file can be used more than once in a single document. The exercises in this book are an example of this.

6. If your files represent logical segments, such as chapters, it is much easier to rearrange your material if you change your mind about the sequence of chapters. If the document were all one file, you'd actually have to move the text to reorganize the document. With this book, when we changed our mind about the sequence of chapters (which we did quite often), all we had to do was change the sequence in which the files were processed.

   *In particular, it is a good idea to put your major figures in separate files, since this makes it much easier to move them around when you are fine-tuning your document.*

Creating a document from more than one input file is easy. All you have to do is use the SCRIPT/VS control word .IM (for *imbed*) and the name of the file you want imbedded. For example, using CMS,

```
.im ss$lists
```

is the control word line we used in this book to imbed the chapter on lists (filename SS$LISTS). Because .IM is a control word, like all SCRIPT/VS control words it must start with the period in column 1.

How you name the file you want imbedded on the .IM control word can vary depending on your computing system and how the file you want imbedded is identified in that system. Consult the document administrator or your supervisor for details on how you should specify the name for the file you want imbedded.

Like ID attributes, it is a good idea to give your files meaningful names, rather than names like CHAPTER1, CHAPTER2, and so on, because updating your document can make names like these useless for identifying what's in them.

A typical way to construct a large document is to have a base file (called the *primary input file*) that consists of:

- The major element tags that form the structure of the document
- Imbed controls for the files that actually contain the text
- Comments that explain what the files contain.

[20] What we call "files" here are also known (in TSO MVS and VSE) as "data sets" and (in ATMS-III) as "documents." For our purposes, these terms all mean the same thing.

For example, a portion of the base file for this book, which was developed using CMS, looks like this:

```
.
.
.
.*
 SSAPPXA is creating and editing a file
.im ssappxa
.*
 SSAPPXC is solutions for exercises
.im ssappxc
.*
 SSAPPXD is differences from earlier releases
.im ssappxd
.
.
.
```

As you can see, we started all our filenames with "SS" so we could distinguish them from the files that go into other books, and we used names that help us to remember what is in the file.

The lines beginning with ".*" are called "comment lines." Comment lines are a way of writing notes to yourself (or to other people who may need to look at or perhaps update your source file); they show up in the source file, but they never appear in the formatted output. When SCRIPT/VS sees a line starting with a period in the first position and an asterisk in the second, it ignores anything on that line and goes on to the next line.

Imbedded files can also have other files imbedded in them. For instance, in this book, SS$FM (front matter) imbeds the file for the title page, called SS$TITLE. There is an "imbed trace" included with the cross-reference listing at the back of this book, following the index. If you take a look at that, you'll see it lists the files that were used to create this book, in the order that they were imbedded.

## Exercise: Overall Document Structure

This time we want you to create a document (call it EXERALL) on your own. Create the front matter and the body. Put in at least one level 1 heading in the body, and imbed the other exercise files you have created.

If you like, you might put some of the exercises in as appendixes. Don't forget to put in the GDOC end tag.

**Note:** You can use either the FPASSES(**n**) (where **n** is the *number* of passes specified) *or* the TWOPASS option of the SCRIPT command to produce correct formatting.

To see the results back at your terminal (in single column rather than the *offset style* we show you here):

**TSO users enter:**[21]

```
script exerall prof(dsmprof3) cont twopass
```

EXERALL and DSMPROF3 are assumed to be qualified with your TSO userid and TEXT.

**CMS users enter:**[21]

```
script exerall (prof(dsmprof3) cont twopass
```

EXERALL and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[21]

```
script * prof(dsmprof3) cont twopass
```

The document to be formatted (EXERALL—referred to by the * in the command) is in your working storage. DSMPROF3 is in SYSOP permanent storage.

[21] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

# Chapter 9. Indexing

Indexing is a bit more complicated than using the other starter set tags. You might want to skip over this section until you have a need to create an index.

Creating an index using GML is somewhat like creating the table of contents. SCRIPT/VS builds the table of contents for you from the items in the text that you have tagged as headings. Similarly, it will build an index for you from items in the text that you have tagged as index entries. Just as you use the TOC tag to show that you want a table of contents, you use the INDEX tag in the back matter of your document to show that you want the index included.

The main difference between the two is that the headings are also printed in the body of the document (where you entered them), whereas the items you tag as index entries appear only in the index (sorted for you by SCRIPT/VS, of course).

By putting the index entries in the text at the point where the information you are indexing occurs, you can ensure that, just as with the table of contents, the page references in the index will be correct whenever you format your document.

Never again will you have to update an old-fashioned index by looking up and changing the page numbers!

Also, you can tell from looking at the source document whether something is indexed or not. This gives you a means of checking the quality of your index.

Figure 7 on page 90 illustrates the terminology we use in this chapter to describe the elements of an index.

**Note:** The indexing tags are acted upon *only when the INDEX option is specified in the SCRIPT command* (as described in "Instructions to SCRIPT/VS" on page 119); otherwise, they are ignored. This is to save the processing time required for the index when an index is not needed—for instance, on drafts.

## *Creating Index Entries*

The simplest kind of index entry, a primary entry with a page number, is entered with the I1 tag, which says, "This is an index subject at the first level." It would look like this:

```
:i1.sauces
```

However, most indexes run to more elegant structures, with primary and secondary and sometimes tertiary entries.

A primary entry that has secondary entries may or may not also have page references of its own, as may secondary entries that have tertiary entries. The primary and secondary entries that have no page references associated with them are called "index entry headings."

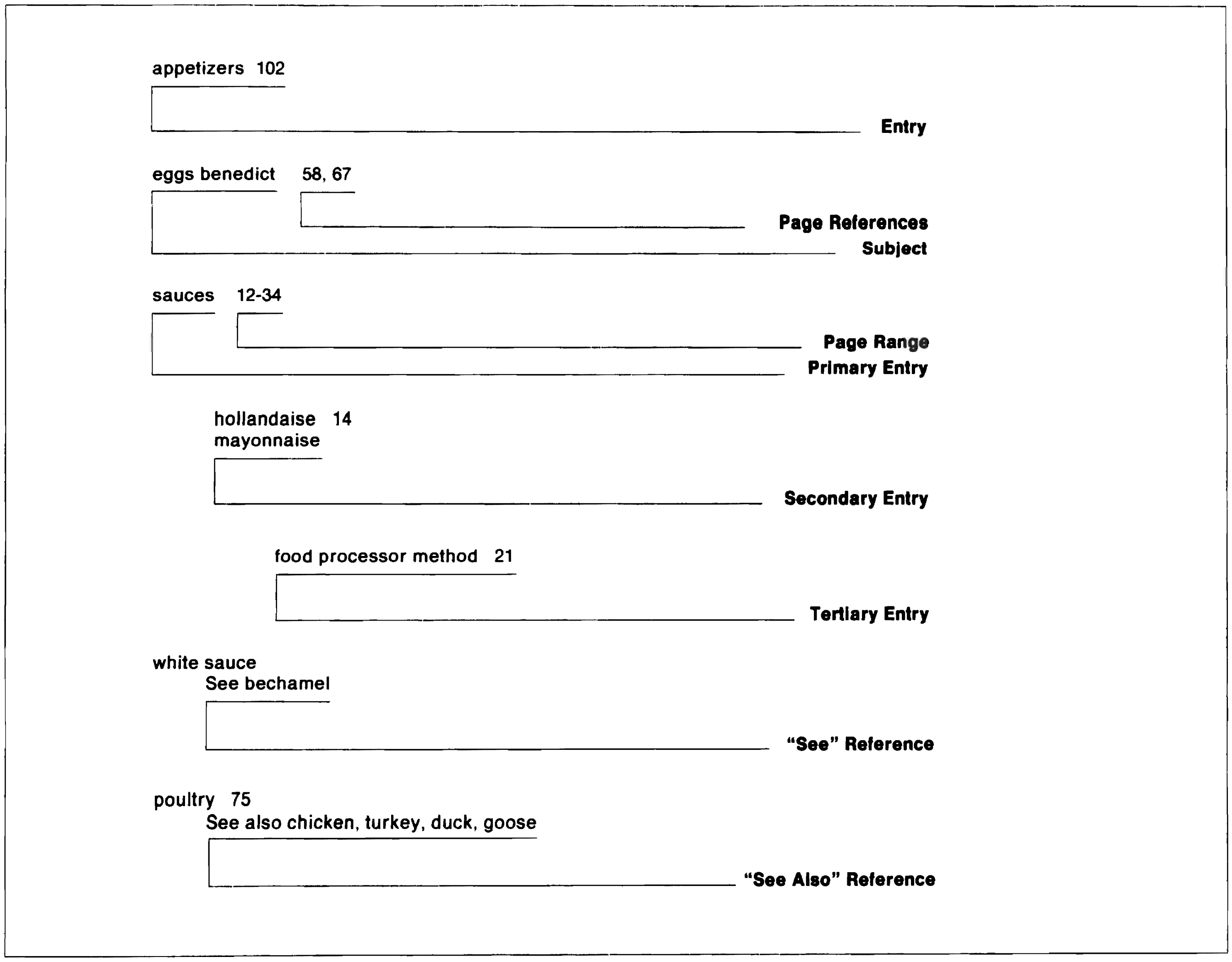


Figure 7. Terminology Used in Discussion of Indexing

There are two ways to associate secondaries with their primaries and tertiaries with their secondaries:

- By their position in the source file, or
- Through use of the ID and REFID attributes.

We'll tell you all about the position way first.

Our index entry:

```
:il.sauces
```

will, as we mentioned, give us a primary subject with a page number.

Elsewhere in our document, we may want some subentries under "sauces," but we don't want another page reference added to the "sauces" line. We could do it like this:

```
:ihl.sauces
:i2.hollandaise
.
.
.
:ihl.sauces
:i2.bearnaise
.
.
.
:ihl.sauces
:i2.bechamel
```

The IH1 tag says, "this is a primary subject with no page number."—in other words, an index entry heading. The I2 tag says, "this is a secondary subject with a page number."

When SCRIPT/VS processes the index entries, it matches all the primaries, regardless of whether they were entered with I1 or IH1 tags, which is just what you want it to do. Thus, our entries so far would give us this result:

```
sauces  12
   bearnaise  14
   bechamel  17
   hollandaise  14
```

If you didn't want a page reference on the "sauces" entry, you would just leave out the I1 tag, and you'd get:

```
sauces
   bearnaise  14
   bechamel  17
   hollandaise  14
```

If you didn't want to say exactly which sauces are where, you would just enter:

```
:i1.sauces
.
.
.
:i1.sauces
.
.
.
:i1.sauces
.
.
.
:i1.sauces
```

and you would get:

```
sauces  12, 14, 17
```

Since both hollandaise and bearnaise are on page 14, SCRIPT/VS has eliminated the duplicate page reference for you.

The same general principle applies to the tertiary level as well. For instance, these tags:

```
:ih1.sauces
:ih2.mayonnaise
:i3.hard way
.
.
.
:ih1.sauces
:ih2.mayonnaise
:i3.blender method
.
.
.
:ih1.sauces
:ih2.mayonnaise
:i3.food processor method
```

added to our existing tags would give us:

```
sauces  12
   bearnaise  14
   bechamel  17
   hollandaise  14
   mayonnaise
      blender method  20
      food processor method  21
      hard way  18
```

If the food processor method and the blender method are discussed together, you can cover them both in a single set of entries, like so:

```
:ih1.sauces
:ih2.mayonnaise
:i3.food processor method
:i3.blender method
```

When the ID/REFID attributes are not used, and the index entries are purely positional, the I3 is associated with the last level 2 entry (either I2 or IH2) that occurred, no matter what has come between (except an I1 or IH1, in which case you have a markup error). The same rule applies to associating secondaries with primaries.

In fact, much text could have intervened between our IH2 in the above example and the two I3s. However, we do not recommend that you do this, since, in a later update to the document, some new entries might be inserted between the two, and then your subentries could get tied up with chicken soup instead of mayonnaise.

As you can see from the above examples, repeating the entire structure of primary and secondary in order to enter a tertiary entry can get to be pretty tedious. For this reason, the starter set has the ID and REFID attributes on the indexing tags to allow you to get at the structure with just the name you put on the ID. You will remember ID and REFID from our discussion of cross-references; they work in much the same way with the indexing tags.

When you put an ID attribute on an I1, I2, I3, IH1, IH2, or IH3 tag, the starter set "remembers" *that entry and any higher level entries associated with it.*

For example, if you had these entries:

```
:ih1.sauces
:ih2 id=mayo.mayonnaise
```

The starter set associates the ID "mayo" with both mayonnaise *and* sauces.

Then you can enter:

```
:i3 refid=mayo.hard way
.
.
.
:i3 refid=mayo.blender method
.
.
.
:i3 refid=mayo.food processor method
```

and get exactly the same results we showed you earlier.

When you use the REFID attribute on the I2 and I3 tags, they pick up the *higher levels* needed from the structure identified by associating the REFID value to the ID with the same value. (You can't use REFID on an I1 tag, because there are no higher levels.)

If you want to pick up *all* the levels (that is, you have an identical structure to the one identified with the ID attribute), you use the IREF tag. Since you are picking up all the levels, the IREF tag doesn't need a level indicator of its own.

So if we had many different ways of making mayonnaise with a blender, we could enter:

```
:ih1.sauces :ih2.mayonnaise :i3 id=mayobln.blender method
.
.
.
:iref refid=mayobln.
.
.
.
:iref refid=mayobln.
```

and we would get this result:

```
sauces
   mayonnaise
      blender method  20, 23, 26
```

## Page Ranges—And Other Things You Can Do with Page References

If you want to show that the sauces section covers a lot of territory, you can do it with the PG attribute. PG can be used on I1, I2, I3, and IREF.

To use it to show a page range, you enter:

```
:i1 pg=start.sauces
.
.
.
:i1 pg=end.sauces
```

and you will get:

```
sauces 12-34
```

The starter set pairs the value "start" with the value "end" on a matching entry at the same level and produces the range of pages.

This can also be done (and more easily when you have a structure of more than one level) with the IREF tag, like so:

```
:ih1.sauces
:i2 id=mayo pg=start.mayonnaise
.
.
.
:iref refid=mayo pg=end.
```

which would give you:

```
sauces
   mayonnaise  18-26
```

If you want to point your reader to the most important reference in a string of page references, you can use the PG attribute to say this is the "major" entry. When you tell the starter set that a reference is major, it puts it first in the string of references. The markup might look like this:

```
:i3 refid=mayobln pg=major.blender method
```

The other thing you can do with the PG attribute is specify some text you want used in place of a page number. It might look like this (since our document is so big it takes two volumes):

```
:i1 pg='(Volume II)'.marinara sauce
```

and it would give you:

```
marinara sauce  (Volume II)
```

## See and See Also

Another common need in indexes is for "see" and "see also" references.

For this, the starter set gives you the SEE and SEEID attributes on the IH1, IH2, and IREF tags. SEEID is like REFID—it points to a name given in an ID attribute. SEE allows you to specify the text you want in the reference.

You should use SEEID whenever possible, because it ensures that the entry you are pointing your reader to in the index exists. (You'll get a message in your cross-reference listing if it doesn't.) But sometimes that's not feasible, so you have the SEE attribute also.

Your markup would look like this:

```
:i1 id=bech.bechamel sauce
.
.
.
:ih1 seeid=bech.white sauce
```

Now if white sauce doesn't have any page references of its own (no I1 tag for white sauce anywhere) and also doesn't have any secondary entries, the above markup will give you:

```
white sauce
   See bechamel sauce
```

But if there were other white sauce entries, then you would get:

```
white sauce  32, 33
   See also bechamel sauce
```

SCRIPT/VS figures out for you whether the reference should be a "see" or a "see also." You don't have to worry about it!

SEE works just the same, except that you supply the text you want for the reference, like so:

```
:ih1 see='chicken, turkey, duck, goose'.poultry
```

and, assuming poultry has another index entry, you get:

```
poultry
   See also chicken, turkey, duck, goose
   how to buy  56
```

In this case, it's up to you to make sure that chicken, turkey, duck, and goose can all be found in your index.

The "see also" reference is always placed first in the subentries.

When SEEID or SEE is used on an IREF tag, the REFID must point to a primary or secondary entry. Tertiary entries cannot have "see" or "see also" references.

If the SEEID points to an ID on a secondary or tertiary entry, the starter set will construct the full cross-reference for you, like so:

```
:ih1.sauces
:i2 id='vinaig'.vinaigrette
:ih1 seeid='vinaig'.oil and vinegar dressing
```

which would give you:

```
oil and vinegar dressing
   See sauces, vinaigrette
.
.
.
sauces
   vinaigrette  47
```

Most of the time, this will be what you want. If not, use the SEE attribute in place of SEEID and spell out exactly what you want.

## *Sorting Index Entries*

The following list defines the basic sort sequence which is common to all of the IBM-supplied languages. SCRIPT/VS sorts the index entries in this order:

1. Blank (code point 40).
2. Required blank (code point 41), numeric space (code point E1), and characters to be "omitted." (Omitted characters such as the hyphen and the apostrophe do not have unique sort values. They sort as if they were required blanks.)
3. Alphabetic characters including the associated accented characters (sorted alphabetically).
4. Numeric characters (sorted numerically).
5. Special characters (sorted in order according to their code point values).

In addition to the common sorting conventions above, the sort sequence for each language includes the special sorting requirements of that language. See the *Document Composition Facility: SCRIPT/VS Language Reference* for additional information on the sort sequence for each supported language. See your supervisor or the document administrator if you need to modify any of the sort sequences.

There are times when the automatic sort provided by SCRIPT/VS will not be suitable to your needs. Some examples of this are:

- When you want to index titles without regard to leading articles, such as "The" and "An"; for instance, when you want to put "The Wind in the Willows" under "wind."
- When you have index entries that start with a special character, but you want them sorted on the first alphabetic character rather than the special character; for instance, if you wanted to put "$IBSYS" under the I's instead of the $'s.
- When you want to put numeric subjects in the alphabetic section as if they were spelled out; for instance, "8-bean salad" sorted as if it were "eight-bean salad."

The text that goes with an index tag (that is, the text following the markup-content separator) is *always* used for sorting. But you can use the PRINT attribute (on the IH tags) to have something else actually printed in your index. To do this, you do two things:

1. Use an IH1, IH2, or IH3 tag with the PRINT attribute to show what you want printed when an entry with matching text (or a REFID pointing to the IH) is used. The IH tag with the PRINT attribute *must come before* any other index entries you want to appear under the heading created with that PRINT attribute.
2. Use an I1, I2, I3 (same level as the matching IH) tag with the same text as the IH tag, or an IREF tag that points to an ID on the IH tag.

For example, suppose we wanted to put "8-bean salad" in the E's. Our markup would look like this:

```
:ihl id=eight print='8-bean salad'.eight-bean salad
.
.
.
:il.eight-bean salad
.
.
.
:iref refid=eight.
```

The I1 tag and the IREF tag would both have the same result; either one or both could be used. Your output would look something like this:

```
.
.
.
eggs benedict  58
8-bean salad  82, 106
endive, Belgian  75
.
.
.
```

Remember, what you want printed goes with the PRINT attribute. How you want it sorted depends on the text that follows the markup; this is sometimes called the "sort key."

The sort key only needs to be long enough to guarantee that the entry will be sorted as you want. In our example above, all we would really need is:

```
:ihl id=eight print='8-bean salad'.ei
```

since "ei" is sufficient to ensure the sorting sequence we want. We recommend, however, that you make the key somewhat longer, to ensure that you will still get what you want when someone comes along and adds "eiderdown quilts" to this incredible cookbook.

## *Notes on the Normal Sort*

Blanks are always sorted first (unless your organization has modified the index sort sequence). Thus, "egg whites" sorts before "eggs benedict."

When SCRIPT/VS is sorting the index entries, it treats them as if they were entered in uppercase. Thus, if you have two entries that are identical except for the case of the letters (for instance, "time" and "TIME"), SCRIPT/VS will sort them as if they were the same. Their order when printed depends on which one is processed first.

If it is necessary to have them both in your index, use the PRINT attribute as we describe above to control the sort. For example, if you wanted "TIME" to come after "time," you could use tags like this to get the sorting you want:

```
:ihl print='TIME'.timez
:il.timez
.
.
.
:il.time
```

and you would get:

```
time  78
TIME  72
```

since "timez" sorts after "time".

## Where To Put Index Entries

Your index entries can be entered just about anywhere; they don't cause any variation in how the text around them is formatted. Here is a list of good places to put index entries to ensure the proper page reference:

- Immediately following a heading.
- Following the first input line of a paragraph or paragraph continuation.
- Following the first input line of a list item or definition description.
- Immediately following an XMP tag, FIG tag, or TABLE tag. (in particular, if you are indexing a figure that is going to float, the index tag must be inside the figure).

Because the starter set keeps, for example, the first few lines of a paragraph on the same page, using the index tags in these places ensures that the page reference picked up for the entry is the same page on which the paragraph starts. If the index entries were placed before the paragraph, they might be processed (and the page number picked up) before the starter set discovers that it has to start a new page for the paragraph.

It is a good idea to put index entries that don't have page references associated with them (see and see also, index headers with no immediate subentries with page numbers, index headers with the PRINT attribute) in one place in the front of your source document; if you scatter them through your document, you will have trouble finding them when you want to change them, since the index itself won't give you a page number to help you find them.

## *Getting the Index*

The INDEX tag shows where you want your index placed in the back matter.

It would look like this:

```
:backm.
.
.
.
:index.
:egdoc.
```

and that's all there is to it, except that you must use the INDEX option on your SCRIPT command.

When you don't need the index for a particular run, you omit the INDEX option on the SCRIPT command, and you save the processing time involved in sorting and formatting the index. (Your INDEX tag will give you a heading for the index, but the index will be empty.)

Running headings and footings will be suppressed on pages following the index.

To get an idea of what your final index will look like, just look at the index in this book; it was done using these tags.

## *Summarizing the Indexing Tags*

At this point, you're probably having trouble keeping track of what attributes go on which indexing tags, so we will summarize them for you here. First we'll show you the tags; then we'll discuss the attributes.

**Index Entry Term (I1, I2, and I3)**

```
:Ix          (x is 1, 2, or 3)
      ID=name
      PG=page reference
      REFID=name.                   (I2 and I3 only)
```

The text of the index entry must be either all on the same line as the end of the markup or all on the next line.

**Index Entry Heading (IH1, IH2, IH3)**

```
:IHx         (x is 1, 2, or 3)
      ID=name
      SEEID=name                    (IH1 and IH2 only)
      SEE='text for see/see also'   (IH1 and IH2 only)
      PRINT='text to be printed'.
```

The text of the index entry heading must follow the same rule as for index entries.

**Index Entry Reference (IREF)**

```
 :IREF
      REFID=name                  (required)
      PG=page reference
      SEEID=name
      SEE='text for see/see also'.
```

There is no text associated with the IREF tag.

**Index**

```
 :INDEX.
```

There is no text associated with the INDEX tag.

**Attributes:**

All of the attributes are optional except for the REFID attribute on the IREF tag.

**ID** = name

This attribute gives a name to this level of index subject and all higher levels associated with it. The name can be up to seven characters long (letters and numbers; the first character should be a letter).

**REFID** = name

This attribute is used to refer to an index structure of the same name, identified with an ID attribute. When used on the I2 or I3 tag, it refers to the levels higher than the level of this tag; when used on the IREF tag, it refers to all the levels in the structure named.

REFID is required for the IREF tag.

**PG** = page reference

Used on the I1, I2, I3, and IREF tags, this attribute allows for a page reference other than just the page number. The value can be any one of these things:

| | |
|---|---|
| **MAJOR** | Identifies this entry as the major entry in a string of entries. DSMPROF3 will put the page number of this entry first. |
| **START** | Identifies this entry as the start of a page range. |
| **END** | Identifies this entry as the end of a page range. |
| **'string'** | Identifies a string of characters (text) that is to be used in place of a page number; it must be enclosed in single quotation marks. |

**SEEID** = name

This attribute identifies the entry with a matching name in its ID attribute that is to be used as the text for a "see" or "see also" entry.

**SEE** = 'text for see/see also'

This attribute identifies the text (enclosed in single quotation marks) that is to be used for a "see" or "see also" entry.

**PRINT** = 'text to be printed'

The text following the end of the markup content separator (.) is the text that is *always* used for sorting the entry.

This attribute is used on the index header tags to identify the text (enclosed in single quotation marks) that is to be printed in the index in place of the text of the entry that follows the end of the markup. When it is used, it must come before any other use of the entry.

## *Readable Source Files*

Before we leave the topic of indexing, we want to tell you one more thing.

Where documents are heavily indexed, the index entries in the source file can make the source file hard to read when you are updating it. You might find it desirable, especially where you have big blocks of index entries (there are some places in this book where the index entries take up 20 or more lines at a time), to set them off with comment lines around the block.

If you'll recall, a comment line is one that begins with a period and an asterisk in the first two positions of the line. When SCRIPT/VS sees that, it ignores the rest of the line, so you can put anything else on it that you please.

So you could, if you want to, enter lines like this around your index entries:

```
.* * * * * * * * * * * *
.* * * * * * * * * * * *
(block of index entries)
.* * * * * * * * * * * *
.* * * * * * * * * * * *
```

if you find that helps you to read your source file.

## Exercise: Indexing

Now create your own index for EXERALL (the composite file you created earlier), so that you get a feel for how the indexing tags work. (Put the index entries into your EXER1-EXER5 files.) Don't forget to put a BACKM tag and an INDEX tag in EXERALL, just before your GDOC end tag.

**Note:** You can use either the FPASSES or TWOPASS option of the SCRIPT command to produce correct formatting.

To see the results back at your terminal:

**TSO users enter:**[22]

```
  script exerall prof(dsmprof3) co twopass index
```

EXERALL and DSMPROF3 are assumed to be qualified with your TSO userid and a type of TEXT.

**CMS users enter:**[22]

```
  script exerall (prof(dsmprof3) co twopass index
```

EXERALL and DSMPROF3 are assumed to have a filetype of SCRIPT and to be on a disk to which you are linked and accessed.

**ATMS-III users enter:**[22]

```
  script * prof(dsmprof3) co twopass index
```

The document to be formatted (EXERALL—referred to by the * in the command) is in your ATMS working storage. DSMPROF3 is in the SYSOP (system operator) permanent storage.

[22] Check with your supervisor or the document administrator to see if there are any special requirements for specifying DSMPROF3 in your organization.

# Chapter 10. Process-Specific Controls

Our last tag is PSC, for *process-specific controls.*

The name of this tag is a little alarming, compared to the other tag names in the starter set. Here's what it means:

Between the PSC tag and its end tag, we are going to include controls (such as SCRIPT/VS control words) that are intended for a *specific* process, rather than the *generalized* markup in the rest of the document. The intent with generalized markup is that it can be processed for any intended use—for instance, any output device that is supported.

Here's how PSCs are used:

- For cases where the formatting produced by the starter set is not satisfactory (for instance, when formatting breaks text at a bad place); we call these "patch" elements, because they are generally temporary until the document is updated.

  In many cases, patch elements depend on the device being used. The PSC tag has an attribute that allows us to say, "Do this only when formatting for this particular device, with these characteristics."

  For example, if you put in a patch to, say, force a column eject when printing on the 1403 at 8 lines per inch, you wouldn't want the same column eject when printing at 6 lines per inch; PSC allows you to control whether or not the column eject is processed.

- For document elements for which there are no GML tags available; we refer to these as *graphic* elements. Graphic elements are very often required in the body of figures and can be charts, diagrams, pictures, or any kind of image.

  This case, like the patch elements, can also depend on the output device characteristics.

- For special purposes or applications as defined by your organization.

  For example, it is possible to use SCRIPT/VS to process GML and produce output that is, rather than a formatted page, a file containing the processing controls for another use of the text. An example of this would be to use SCRIPT/VS to produce processing controls for a second text processor that formats text for a photocomposer. In such cases, it is sometimes necessary to include direct controls for that second text processor. Here, we would use PSC tags to tell SCRIPT/VS to pass these controls directly to the output file without doing any kind of processing on them.

  When you use SCRIPT/VS to format this text directly (instead of producing controls for the second text processor), the PSC tag tells SCRIPT/VS to ignore these controls.

The PSC tag has a PROC attribute, which is how you specify the circumstances under which the lines between the PSC and its end tag are to be processed. In the PROC attribute, you can specify one or more physical or logical device types and/or one or more "conditions."

## Physical Devices

You specify a *physical device* (the actual piece of equipment or hardware—such as a 3270 or a 3800) when you are concerned with what type of output device is used, but you don't care about the special characteristics of that output device. The physical devices you can specify on the PROC attribute are:

| | |
|---|---|
| 3800 | The IBM 3800 Printing Subsystem Model 1 |
| 1403 | The IBM 1403 Printer or other printers that look the same to SCRIPT/VS |
| 3270 | The IBM 3270 Display terminal family |
| 2741 | The IBM terminal that looks like a typewriter |
| 4250 | The IBM 4250 printer |
| 38PP | IBM 3800 Printing Subsystem Model 3 and Model 6 |
| 3820 | The IBM 3820 Page Printer. |

## Logical Devices

*Logical devices* are a bit more involved, because they take into account not only which piece of equipment you want your output to go to, but also such things as the lines per inch and the size of the paper. Here's a sample of a few of the logical devices you can specify:

| | |
|---|---|
| 3800n8 | The 3800 at 8 lines per inch with narrow paper. |
| 1403w6 | The 1403 at 6 lines per inch with wide paper. |
| term | The terminal. |

A complete table of logical device types is given in Figure 8 on page 125.

Now, if you had this PSC tag sequence as part of your document

```
:psc proc='3800n8'.
(specify control words here)
:epsc.
```

the control words between PSC and its end tag would be processed only if "3800n8" were specified as the logical device on the SCRIPT command when you requested formatting for your document.

## Conditions

*Conditions* are simply names you make up (eight letters or numerals; the first character should be a letter). A condition specified in the PROC attribute is considered to be "true" when that condition name is paired with the SYSVAR P in the SCRIPT command to format the document. (SYSVARs are explained in "Instructions to the Starter Set (SYSVAR)" on page 116.)

The PSC tag looks like this:

```
:psc proc='value1 value2 ...'.
.
.
.
:epsc.
```

where *value1*, and so on, are the conditions under which the lines between this PSC tag and its end tag are to be processed. If any of the conditions is *true*, the lines will be processed. Otherwise, they'll be ignored, the starter set will skip to the PSC end tag and go on from there.

To make a condition true and have the control words processed, the SCRIPT command line would have to include this SYSVAR option as part of the command:

```
sysvar (p value1)
```

That SYSVAR option says to SCRIPT/VS, "format the control words between the PSC tag and its end tag that have the 'PROC=value1' attribute and value specified as part of the PSC tag".

If you don't specify

```
sysvar (p value1)
```

any control words or text included within the PSC tag identified by "value1" and its end tag will be ignored.

As we mentioned, conditions that you name yourself are considered to be true when you specify the PROC attribute value with the SYSVAR P in your SCRIPT command. Because SCRIPT/VS knows what physical and logical device type it is formatting for, SCRIPT/VS knows which of those conditions is true.

Getting back to how PSC is used, here's an example of a case where you want a page eject on the 1403 at 6 lines per inch (and narrow paper) and on the terminal, but not on the 3800, 4250, 3820, or the 1403 at 8 lines per inch. Your markup would look like this:

```
:psc proc='1403n6 term'.
.cm page break for 1403n6 or terminal format
.pa
:epsc.
```

The .PA (page eject) control word will be processed only when formatting for the terminal or for the 1403 at 6 lines per inch on narrow paper.

Notice that we used a comment to remind ourselves (or the next person who has to update this document) what the PSC is all about. In this case, we used the .CM (comment) control word instead of the ".*" we normally use for comments. The reason we do this is so that your organization can, if it wants to, provide special processing for the comment. (It couldn't do this if you used the ".*", because comments entered that way are not processed.) Don't put any semicolons on the .CM line; semicolons are special characters to SCRIPT/VS.

If your document had controls in it for some other text processor, such as TERMTEXT, your markup might look like this:

```
:psc proc='ttxt'.
.cm graphic for TERMTEXT processing only
  (TERMTEXT controls)
:epsc.
```

Again, we've used .CM, this time followed by the word "graphic," to remind us of any special processing for graphics.

The PSC tag can also be used without the PROC attribute. In such a case, the starter set doesn't actually do anything with the PSC and its end tag—the lines are processed as if the PSC weren't there.

However, the existence of the PSC tag in your markup allows your organization to provide its own processing for the PSC tag, either now or in the future. Documents marked up with SCRIPT/VS control words that are enclosed in PSC tags can be processed with some other processor where the SCRIPT/VS control words are not recognized.

This use of PSC is to extend or modify the processing provided by the profile (that is, DSMPROF3). A common use of PSC elements is to set your own symbols, which we describe in "Chapter 11. Symbols" on page 109.

The markup would look like this:

```
:psc.
.cm symbols for names
    (SCRIPT/VS control words to set symbols)
:epsc.
```

You should consult your supervisor or the document administrator to find out what the rules in your organization are for entering SCRIPT/VS control words (that is, whether or not your organization requires that you use PSC tags around them). However, in the absence of any specific instructions to do otherwise, you should use PSC tags for every occurrence of control words (except .IM) to ensure the generality of your document and your text data base.

**You must be careful when using SCRIPT/VS control words that they don't interfere with the control words that make the tags work.** See "Symptom: Unexpected Formatting Results" on page 127.

# *Part Two: Formatting Your Document*

In addition to the GML tags themselves, there are a number of other things you'll want to know about using the starter set.

# Chapter 11. Symbols

A symbol is a name in a source document that can be replaced with something else during formatting.

A common use of symbols is to specify characters that can be printed on the printer but that possibly can't be entered at your keyboard. An example might be the square bullet, which is not available on all keyboards, and which looks like this: □

Because symbols for this purpose vary from organization to organization, the starter set does not include them. However, your organization may have set up symbols for you to use; check with the document administrator or your supervisor if you need to enter characters that don't appear on your keyboard. We mention them here, even though the starter set doesn't include them, so that you'll know how to enter them if your organization provides them.

All symbols are entered the same way: with an ampersand (&), followed by the symbol name, followed by a period. So a symbol for the square bullet that is named "sqbul" would look like this:

```
&sqbul.
```

(Don't try to use this particular symbol unless you have checked and know it is available in your organization.)

The period here is just like the period you use at the end of your GML markup; it won't print in your output document.

As we mentioned earlier, all your GML tags can be entered in either uppercase or lowercase. We recommend, however, that you always use lowercase.

***This is not true for symbols***! Symbols are what we call "case-sensitive"; that is, the symbol &ABC. is a different symbol from the symbol &abc. This means that when you use symbols, you must know the case of the name as well as the spelling.

## *Symbols Provided by the Starter Set*

The starter set provides these symbols:

| | |
|---|---|
| **&rbl.** | required blank |
| **&gml.** | GML delimiter |
| **&amp.** | ampersand |
| **&semi.** | semicolon |
| **&date.** | date |
| **&time.** | time |

Here's what these are for:

**&rbl.** Required blank.

This symbol is used when you want to force SCRIPT/VS to give you a blank under conditions where it otherwise might not because it is formatting the text. For instance, &rbl. will prevent SCRIPT/VS from splitting a line between two words when formatting. For example, if you were formatting a paragraph with a number in the international style (blanks instead of commas), you could enter the number like this:

```
1&rbl.000&rbl.000
```

and SCRIPT/VS would keep it all on one line with just one blank where each &rbl. was requested. It would look like this:

```
1 000 000
```

Using &rbl. will also prevent SCRIPT/VS from inserting extra space between two words when it is justifying text. For instance, you could enter:

```
2&rbl.x&rbl.4
```

to ensure that the space on either side of the "x" is only one space wide.

The &rbl.'s would ensure that your output would look like this:

```
2 x 4
```

This symbol can also be used to insert blank lines in figures and examples; just put the &rbl. symbol on the line by itself.

And there are some cases in figures and examples where SCRIPT/VS will take out extra blanks; if you replace the first blank with an &rbl. symbol, they won't get taken out.

**&gml.** The GML delimiter.

You only need to use this symbol when you are writing a document about the GML tags.

In normal text, all colons are printed exactly as they are entered. However, if the colon is followed by a valid GML tag name, SCRIPT/VS will read it as a tag. Thus, if you are writing a book about GML tags and you want to illustrate a tag, you can't enter it with a colon. Instead, you must use the &gml. symbol. This book is loaded with examples of tags that were entered this way:

```
&gml.tagname.
```

**&amp.** The ampersand.

This symbol is similar to the &gml. symbol above, except that it is used when you want to illustrate a valid symbol.

Like the colon, ampersands occurring in normal text will print as ampersands. However, if they are followed by a valid symbol name, they will be considered as symbols and the appropriate substitution will be made. If you are writing a document about valid symbols, then you will need to use this symbol in place of the &. This chapter has many instances of this kind of entry:

```
&amp.symbolname.
```

In fact, to illustrate the symbol "&amp." itself, we entered:

```
&amp.amp.
```

**&semi.** The semicolon.

The semicolon is used by the starter set as the SCRIPT/VS *control word separator.* Normally, this is not something you are concerned with. However, if you are illustrating the use of control word separators, you will need to use this symbol in place of the actual semicolon.

Another time when you will need the &semi. symbol is when you want to use a semicolon in your own symbol values, as described in the next section, or in the value of an attribute.

There can also be cases where SCRIPT/VS will misinterpret a semicolon as the control word separator and not print it; simply replace that semicolon with the &semi. symbol, which always prints.

**&date.** The date.

When you want to use the date of the document in places other than the title page, you use the symbol &date. If the DATE tag was entered with a date, then that date is printed when you use this symbol. Otherwise, the system-supplied date of processing is printed.

For example, if you enter the following sentence in your source document:

```
:p.This document was printed on &date..
```

it would appear in your output like this:

This document was printed on February 26th, 1987.

**&time.** The time.

Use the &time. symbol whenever you want the system-supplied current time to appear in your document.

For example, if you enter the following sentence in your source document:

```
:p.This document was processed at &time.
```

it would appear in your output like this:

This document was processed at 9:02 a.m.

## *Your Own Symbols*

You can set up symbols for your own use, to keep from having to enter the same lengthy phrase over and over. Symbols are also useful if the name of something you're writing about is likely to change. If you define a symbol name and then use the symbol name in the text, you will only have to change the symbol value when the name is changed.

Here's how you define a symbol name:

```
  .se symbolname = 'text you want substituted'
```

The .SE is the SCRIPT/VS "Set Symbol" control word and must start with the period in column 1. The *symbolname* can be anything you like, up to ten characters long (letters and numbers only). The *text you want substituted* (the symbol value) is the text that will appear whenever you use *&symbolname.* in your document. The symbol value must be enclosed in single quotation marks if it contains any blanks or special characters, such as punctuation characters. If it has a single quotation mark within it, then you must use two single quotation marks in a row, so SCRIPT/VS will know that you do not intend that single quotation mark to be the end of the symbol value.

So you could define a symbol like this at the beginning of your document:[23]

```
  .se dcf = 'Document Composition Facility'
```

and then everywhere you used &dcf. in your document, like so

```
  :p.SCRIPT/VS is the formatter component of the &dcf..
```

you would get:

SCRIPT/VS is the formatter component of the Document Composition Facility.

Notice that you *do not use the & in the .SE control word, but you must use the & when you actually use the symbol.* Also notice that there are two periods at the end of the markup: the first period ends the symbol, the second period ends the sentence.

Here's an example of how to specify a symbol value that uses a symbol name and has a single quotation mark in it:

```
  .se dcfpg = '&dcf.: SCRIPT/VS Text Programmer''s Guide'
```

We used two single quotation marks together (*not* a double quotation mark) in "Programmer's" so SCRIPT/VS would know we didn't intend that quotation mark to be the end of the symbol value.

So now we could say:

```
  :p.Consult the :cit.&dcfpg.:ecit.
  for more information on the &dcf..
```

and we would get:

Consult the *Document Composition Facility: SCRIPT/VS Text Programmer's Guide* for more information on the Document Composition Facility.

[23] Of course, follow the rules in your organization for using SCRIPT/VS control words in your document; see "Chapter 10. Process-Specific Controls" on page 103.

If we had slipped up and typed &DCFPG. and &DCF. instead (forgetting that we had defined our symbols with lowercase names), we would get:

---

Consult the *&DCFPG.* for more information on the &DCF..

---

So now you know what happens if you enter a symbol name incorrectly—SCRIPT/VS treats it as text. If you follow the rule of *always* using lowercase letters for symbols, IDs, and so forth, then you never have to worry about remembering what case they are in.

**Long Input Records:** Defining your own symbols is also a way of getting 256 characters worth of information into an input record.

You might have to do this if the editor program or word processing equipment you are using does not allow you to create input records that long, but you need to enter long titles, headings, or figure captions.

For example, the title of this book could have been done like this:

```
.se dcf = 'Document Composition Facility'
.se gmul = 'Generalized Markup Language'
.se ssug = 'Starter Set User''s Guide'
.se stpg = '(Sample Title Page)'
```

and we could then enter our TITLE tag like so:

```
:title stitle='GML &ssug.'
:title.&dcf.:  &gmul. &ssug. &stpg.
```

which is short enough to fit anywhere.

# Chapter 12. Things You Can Specify At Run Time

We've mentioned many times in this book things that you can specify when you request formatting for your document.

Basically, there are two types of instructions you can specify at run time:

1. Instructions to the starter set, which specify such things as:

   - Whether or not H0 and H1 force an odd-numbered page (default is no odd-numbered page)
   - Whether or not headings are to be numbered (default is no head level numbering)
   - Whether the document is to be

     a. Single column (default);

     b. Double column; or

     c. Offset style (looks like this book)

   - Whether or not a title page is to be printed, and how it is supposed to look (default is like the sample title page in this book)
   - Whether or not a cross-reference listing is to be produced (default is yes)
   - A value to test for in the PROC attribute of the PSC tag (as discussed in "Chapter 10. Process-Specific Controls" on page 103).

   These instructions are specified as "SYSVARs" (system variables) in the options on the SCRIPT command or in an options file called by the SCRIPT command.

2. Instructions to SCRIPT/VS itself, which specify such things as:

   - The profile to be used (in our case, DSMPROF3)
   - The size of the left-edge margin
   - The device that formatting is to be done for
   - Whether or not the output is to be printed, displayed, or written to a disk file
   - Whether or not the index tags are to be processed to produce an index
   - Whether or not spelling verification is to be done
   - Whether or not read and write files (SYSVAR R and W) are to be created in order to correctly resolve page references.

   The SCRIPT command options that are most commonly used are described in "Instructions to SCRIPT/VS" on page 119. For a complete description of all the options that can be specified, see *Document Composition Facility: SCRIPT/VS Language Reference*.

## *Instructions to the Starter Set (SYSVAR)*

System variables (SYSVARs) are values that can be passed to the starter set on the SCRIPT command (or, for CMS and ATMS-III, an options file called by that command) that allow you to control certain aspects of the starter set processing.

SYSVARs are specified in your command options as follows:

```
  SYSVAR(c value c value ...)
```

where *c* is the single character designation of the system variable you are specifying and *value* is the value you are assigning to it.

The SYSVARs recognized by the starter set are as follows:

***C*** ***Value and Meaning***

**S** (Style) can be set to:

| | |
|---|---|
| one (1) | single column (default) |
| two (2) | double column |
| OFFSET | offset |

Single column (1) sets both text and headings flush left.

Double column (2) sets two columns.

Offset (OFFSET) is the style used for this publication, where the body and appendix text is set on a line offset from the left margin, with heading levels 0-4 flush left.

As an example, if you want your document formatted in double columns, you would enter:

```
  sysvar (s 2)
```

as an option on the SCRIPT command.

**T** (Title Page) can be set to:

| | |
|---|---|
| RIGHT | title page with elements aligned on the right (default). |
| CENTER | title page with elements centered |
| LEFT | title page with elements aligned on the left |
| YES | title page with elements aligned on the right (same as RIGHT) |
| NO | no title page |

As an example, if you do not want your title page to print, you would enter:

```
  sysvar (t no)
```

as an option on the SCRIPT command.

**D** (Duplexing) can be set to:

| | |
|---|---|
| NO | duplexing off; level 0 and level 1 headings start on the next new page with no blank pages (default). |

| | |
|---|---|
| YES | duplexing on; all level 0 and level 1 headings begin on the next odd-numbered page and are right aligned for one- and two- column formats, leaving blank pages if necessary. |

Duplexing on is intended for documents that will ultimately be printed on two sides of the paper; it causes the running footings to be formatted differently on odd and even pages. Duplexing is requested by the YES option.

Duplexing off (NO) is used when a document is not going to be reproduced on two sides of the paper. The running footings are handled the same on all pages, and there are no blank pages.

As an example, if you want your document printed with duplexing on, you would enter:

```
sysvar (d yes)
```

as an option on the SCRIPT command.

**W** (Write) can be set to:

| | |
|---|---|
| filename | (up to eight characters) |

This may be any name (up to eight characters) that you want to assign to a file of ID information that is going to be created when the document is processed.

This file, newly created as the result of specifying SYSVAR W, will contain all the IDs specified on the heading, figure, footnote, and list item tags.

For example,

```
sysvar (w myids)
```

creates a file, myids, that contains all the IDs (except the indexing IDs) included in the document being processed.

When this newly created file is read (using the SYSVAR R option), the collected IDs are used to resolve forward references in a document so that page references resolve correctly.

SYSVAR R (for READ) and W (for WRITE) can both be specified using the same name. If, for example, you entered:

```
sysvar (r myids w myids)
```

changes made to IDs or IDs added to the document being processed are included in a new file (specified with the SYSVAR W) while the earlier version (specified with the SYSVAR R) is being read.

**Note:** This SYSVAR is valid *only* in CMS and TSO.

**R** (Read) can be set to:

The name of the file that was created using SYSVAR W.

The named file holds the collected IDs from the document that was processed when SYSVAR W was specified.

If, for example, you read the file by entering:

```
sysvar (r myids)
```

this will resolve forward references to the IDs within the document so that page references resolve correctly on the first pass.

SYSVARs R and W can both be specified using the same name.

When you specify both R and W, changes made to existing IDs or new IDs added to the document being processed are included in the newly written file while the earlier version is being read (See **W (Write)** command option).

Once the ID file has become stabilized and no new ID entries are being made, a document can be formatted with correctly resolved forward references with only one pass necessary when SYSVAR R is used.[24]

You can also format parts of a document and get correctly resolved references to parts not being formatted at the same time. You can do this by specifying SYSVAR R and the file that contains all the IDs that was created when the entire document was formatted previously.

**Note:** This SYSVAR is valid *only* in CMS and TSO.

**H** (Head Numbering) can be set to:

| | |
|---|---|
| NO | do not number headings (default) |
| YES | number headings 1 through 4 in the body |
| value | number headings 1 through 4 in the body starting with this value |

This allows you to process pieces of the document and get the headings numbered as if the whole document were being processed. For example, if the first heading in the piece is an H3, and you want it numbered 3.6.2, you would enter:

```
sysvar (h 3.6.2)
```

**X** (Cross-Reference Listing) can be set to:

| | |
|---|---|
| YES | produce cross-reference listing (default) |
| NO | do not produce cross-reference listing |

DSMPROF3 generates an extremely useful listing at the end of your document that contains information about all the ID and REFID attributes used (HDREF, FIGREF, FNREF, and so on) and an imbed trace. You will definitely want this listing while you are developing your document. However, if you are printing many copies (say, for distribution as review drafts), you will want to suppress this listing. This SYSVAR parameter lets you do that.

If you do not want the cross-reference listing to print, enter:

```
sysvar (x no)
```

as an option on the SCRIPT command.

**P** (Process Condition) can be set to:

| | |
|---|---|
| value | sets *value* on (that is, the condition is true) |

This SYSVAR parameter allows you to set the condition that is to be true when processing PSC tags with the PROC attribute specified. (See "Chapter 10. Process-Specific Controls" on page 103.)

[24] You will need to use either FPASSES or the TWOPASS option to resolve forward index references and to fill in the table of contents and the list of illustrations.

For example, if you had a document with two versions of a figure and the version you want printed changes depending on who you are sending the document to, then you might set up your figures like this:

```
:psc proc='figold'.
...fig tags and text...
:epsc.
:psc proc='fignew'.
...fig tags and text...
:epsc.
```

Then, if you want *fignew* printed rather than *figold*, you would enter the following with your SCRIPT command:

```
sysvar(p fignew)
```

If neither FIGNEW nor FIGOLD were specified, both of the PSC/EPSC groupings would be ignored while processing.

You can specify as many SYSVAR parameters as you need—your SYSVAR option would look like the following if you wanted the file named myids to be read to resolve IDs, the figure identified by PSC PROC = FIGNEW printed, with double columns, duplexing on, headings numbered, and no cross-reference listing.

```
SYSVAR(R MYIDS P FIGNEW S 2 D YES H YES X NO)

                                            xref listing
                                            head numbering
                                            duplexing
                                            style
                                            process condition
                                            read/resolve IDs
```

The order in which you enter the pairs does not matter.

## Instructions to SCRIPT/VS

In order to process your documents, you will need to know how to invoke the formatter, SCRIPT/VS. You will also need to be familiar with some of the SCRIPT/VS command options. We discuss some of the commonly used command options later in this chapter. A complete description of the command options is given in the *Document Composition Facility: SCRIPT/VS Language Reference.*

Some of the things we discuss here are independent of the computing system you use and the way in which your organization uses it. These generally deal with the formatting options (such as the SYSVARs we talked about before, whether the index is to be produced, and how many passes you want) and other services SCRIPT/VS provides, such as spelling checking and error message handling.

Others are not; these generally deal with how data sets, files, or documents are identified, what output device specifications can be used, and how output is routed to a device. Your organization may have set up special commands or procedures for you to use, instead of what we show you here, to handle these kinds of things.

### *Identifying the Document to be Processed*

The command that invokes SCRIPT/VS is "SCRIPT".[25] The name of the document to be processed is entered immediately after the command, separated from the command name by at least one blank:

```
   script mydoc
```

If you are using TSO, a data set name that is not fully qualified is assumed to have the default qualifiers of your userid and TEXT.

If you are using CMS, the default filetype is SCRIPT.

If you are using ATMS-III and the document is in your working storage, use an asterisk (*) for the document name.

If you are using any other computing environment, consult your supervisor or the document administrator as to what to do.

The document name is followed by any options to be used. The SCRIPT command options that are most commonly used in GML processing are described below.

## SCRIPT Command Options

The first parameter of the SCRIPT command is an identifier of the file or data set that contains your document. Options follow the document name.

In CMS, options must be separated from the document name by a left parenthesis "(":

```
  script mydoc ( options
```

In TSO and ATMS-III, options are separated from the document name by at least one blank:[26]

```
  script mydoc options
```

Separate options from each other by at least one blank. Some options have values or "sub-options"; enclose these values in parentheses.

In the descriptions below, options are shown in uppercase letters and values are shown in lowercase letters. You may enter the options in either uppercase or lowercase; whichever is most convenient for you.

### *Processing Options:*

**BIND(obind ebind):**

Specifies that the printed portion of the output page is to be shifted to one side to leave room for binding. It is shifted "obind" space units to the right on odd pages, and

[25] Even this may have been altered by your organization. For instance, if you have more than one release of SCRIPT/VS available, then each one of them has to be called by a different name.

[26] ATMS-III users should refer to the *ATMS-III: Terminal Operator's Guide* for additional information on using SCRIPT/VS in your environment, especially when you use the LIB option of the SCRIPT command or when you want your output sent to an off-line printer.

"ebind" space units on even pages. (You would normally make "ebind" smaller than "obind" for duplex printing, where the output is ultimately to be printed on two sides of the paper, so even pages would have a large enough right margin for binding.)

The "ebind" value may be omitted when you want all pages shifted the same amount (as you normally would for printing on one side of the paper).

If you don't specify any bind at all, the default is two characters for terminals and one inch for all other devices.

### CHARS(font1 font2 ... ):

Specifies the fonts to be used when formatting for the IBM 3800 Printing Subsystem Model 1.

You can specify two uppercase and lowercase fonts for the 3800 Model 1—the first is the normal text font, and the second is the font used for headings, highlighting, and so forth. For example, if you were formatting for the 3800 Model 1, you might specify:

```
  CHARS(GT12 GB12)
```

for "gothic text, 12-pitch" and "gothic bold, 12-pitch," respectively. The fonts available for the 3800 Model 1 are shown in *Document Composition Facility: SCRIPT/VS Language Reference*. Your organization may have added its own fonts to the ones available.

**Note:** In addition to the font specification on the SCRIPT command, a compatible font specification must be given in the job control statements for printing the document on the 3800. Consult your supervisor or the document administrator in your organization to make sure the correct font specifications have been made.

You should also consult the document administrator or your supervisor to see what coded fonts are available when you request formatting for page devices. If CHARS is not specified, a default font is used. See "Font Requirements for Page Devices" on page 126 for more information concerning fonts.

### FONTLIB(libname):

Specifies the font library to be used when formatting for page devices. The font library contains character set information and image patterns of these characters. The font library requested on the SCRIPT command must be available on the system when the formatted document is printed.

The fonts available will vary from one organization to another. Refer to "Font Requirements for Page Devices" on page 126. Also, see the document administrator or your supervisor for the fonts used with the page device at your location.

### FPASSES(n):

Allows you to specify more than two formatting passes on documents. When two passes are inadequate to insure that all page number references are resolved correctly, the FPASSES option may be used to specify more passes. The IBM-supplied maximum number of formatting passes is 4. (This may have been changed, however, at your location. Check with your supervisor or the document administrator.)

The FPASSES option causes SCRIPT/VS to process your input document the specified number of times, and produces output only on the last pass.[27]

**Note:** FPASSES and TWOPASS are mutually exclusive options. See "TWOPASS" on page 124 for related information.

[27] Keep in mind that use of this option will significantly increase the processing time for your document.

**INDEX:**

Causes SCRIPT/VS to create an index from index entries specified in the text of the document.

If you omit the INDEX option, any indexing tags in the document will be ignored. See "Chapter 9. Indexing" on page 89 for information on the indexing tags.

**NOSEGLIB:**

Specifies that no library search is to be made for page segments. Use NOSEGLIB when you have specified page segments within your document with the .SI [Segment Include] control word, but those page segments are not yet within a segment library when you issue the SCRIPT command. This will prevent an error message from being issued.

**OPTIONS(name):**

Specifies the name of a file containing more SCRIPT/VS command options. The options are processed as if they had been entered where the OPTIONS option appears.

**Note:** The OPTIONS option is available only in CMS and ATMS-III.[28] In CMS the file containing the options is assumed to have a filetype of OPTIONS.

**PAGE:**

Allows you to print only the specific pages of formatted output that you want. The PAGE options can be specified in different ways depending on the range of pages that you want printed. You can specify the PAGE option like this:

    PAGE [([FROM] frompage [TO] topage)]

    PAGE [([FROM] frompage FOR n)]

    PAGE [([FROM] page ONLY)]

If no sub-option is given with the PAGE option, you will be asked for the page you want printed (except under ATMS-III, where it is ignored.)

Here are some examples of how you could use this option when you issue the SCRIPT command:

| | |
|---|---|
| **page (from 8 to 16)** | this would print out pages 8 through, and including, 16 |
| **page (from 8 for 2)** | this would print out 2 pages—8 and 9 |
| **page (6 only)** | this would print out only page 6 |

**PROFILE(name):**

Specifies the name of a file you want used as the document profile.[28]

The profile provided with SCRIPT/VS for use with the GML starter set is named DSMPROF3.

**SEGLIB(name):**

Specifies the name of the segment library you want searched when formatting for page devices. SCRIPT/VS searches the segment library for any page segments you have included in your document with the .SI [Segment Include] control word.

[28] ATMS-III users should refer to the *ATMS-III: Terminal Operator's Guide* for additional information on using SCRIPT/VS in your environment, especially when you use the LIB option of the SCRIPT command or when you want your output.

If SEGLIB is not specified, SCRIPT/VS automatically searches the default library for requested page segments. You must specify the SEGLIB option if the page segment you request resides in a different library. Keep in mind that SCRIPT/VS searches only one segment library, either the default library or the one you created, but not both.

The default segment libraries for the IBM 4250 printer are:

| *System* | *Default Library* |
|---|---|
| CMS | SEGLIB(PSEG4250) |
| TSO | SEGLIB(SYS1.PSEG4250) |
| ATMS-III | NOSEGLIB |
| Batch MVS | SEGLIB(PSEG4250) |
| Batch VSE | segments are not supported. |

The default segment libraries for the IBM 3800 Printing Subsystem Model 3 are:

| *System* | *Default Library* |
|---|---|
| CMS | SEGLIB(PSEG38PP) |
| TSO | SEGLIB(SYS1.PSEG38PP) |
| ATMS-III | NOSEGLIB |
| Batch MVS | SEGLIB(PSEG38PP) |
| Batch VSE | segments are not supported. |

The default segment libraries for the IBM 3820 Page Printer are:

| *System* | *Default Library* |
|---|---|
| CMS | SEGLIB(PSEG3820) |
| TSO | SEGLIB(SYS1.PSEG3820) |
| ATMS-III | NOSEGLIB |
| Batch MVS | SEGLIB(PSEG3820) |
| Batch VSE | segments are not supported. |

If you know a page segment does not yet exist or there is no segment library, specify the NOSEGLIB option of the SCRIPT command. SCRIPT/VS will not search for a segment library and no error message will be issued. (See NOSEGLIB option.)

**SPELLCHK:**

Causes SCRIPT/VS to perform spelling verification.

Because spelling verification requires additional processing time (each word is checked), you should use this option when your document is otherwise just about complete. You won't want to use it on every draft unless you feel it is absolutely necessary.

If you request spelling checking when processing with the starter set, everything is checked except the contents of examples, figures, running headings, and running footings. (The words that SPELLCHK locates that are not a part of its spelling dictionary are listed with the error messages. We'll talk about this later in "Chapter 14. Starter Set and SCRIPT/VS Messages" on page 131.)

**SYSVAR(x value ....):**

Specifies system variables that can be used to control the processing of a document.

See "Instructions to the Starter Set (SYSVAR)" on page 116 for information on the SYSVAR options supported by the starter set and how to specify them.

**TWOPASS:**

Causes SCRIPT/VS to process your input document twice, producing output only on the second pass.

TWOPASS is used to resolve forward references (for example, when an automatically generated table of contents or list of illustrations is in the front of a book). Not unexpectedly, this option nearly doubles computer processing time for the document.

**Note:** FPASSES and TWOPASS are mutually exclusive options. See "FPASSES(n)" on page 121 for related information.

## *Logical Device and Output Destination*

**DEVICE (type):**

Specifies the "logical" device for which formatting is to be performed. In this context, "logical" means a combination of physical device type, page size, and number of lines per vertical inch. The logical device types supported in the starter set are shown in Figure 8 on page 125.

**Note:** See the Document Administrator for procedures to get your document printed at your location.

**FILE (name):**

Causes output to be formatted for the specified logical device, but stored in the named file or data set instead of being displayed or printed.

If the name (and its surrounding parentheses) is left out, SCRIPT/VS will create a name for the output document by putting a dollar sign ($) in front of the first seven characters of the name of the input file. Be advised, however, that the file name depends on the device type and the operating environment being used.

**Note:** The FILE option is not available in ATMS-III.

**PRINT:**

Causes output to be formatted for the specified logical device, and printed on the system printer (in CMS and TSO).[29] If no logical device is specified, "DEVICE(1403W6)" is assumed.

**TERM:**

Causes output to be formatted for the specified logical device, but displayed at a terminal instead of on the physical device. If no logical device is specified, "DEVICE(TERM)" is assumed.[29]

[29] ATMS-III users should refer to the *ATMS-III: Terminal Operator's Guide* for additional information on using SCRIPT/VS in your environment, especially when you use the LIB option of the SCRIPT command or when you want your output.

| Logical Device Type | Real Device Type | Lines per Inch | Page Size |  | Margins |  |  | Line Length | Class of Device |
|---|---|---|---|---|---|---|---|---|---|
|  |  |  | Width | Length | Bind | Top | Bottom |  |  |
| TERM | (1) |  |  |  |  |  |  |  |  |
| 2741 | 2741 | 6 | 132 | 11 i | 2 | .5i | .5i | 6i |  |
| 3270 | 3270 | 6 | 204 | 11 i | 2 | .5i | .5i | 6i |  |
| 1403N6 |  | 6 | 8.5i | 11 i |  |  |  |  |  |
| 1403N8 |  | 8 | 8.5i | 11 i |  |  |  |  |  |
| 1403W6 |  | 6 | 13.5i | 11 i |  |  |  |  |  |
| 1403W8 | 1403 | 8 | 13.5i | 11 i | 1i | .5i | .5i | 6i |  |
| 1403W6S |  | 6 | 13.5i | 8.5i |  |  |  |  |  |
| 1403W8S |  | 8 | 13.5i | 8.5i |  |  |  |  |  |
| 1403SW (2) |  | 6 | 8.5i | 11 i |  |  |  |  |  |
| STAIRS |  | 6 | 13.5i | 11 i |  |  |  |  |  |
| 3800N6 |  | 6 | 8.5i | 10 i |  |  |  |  | line |
| 3800N8 |  | 8 | 8.5i | 10 i |  |  |  |  | devices |
| 3800N12 |  | 12 | 8.5i | 10 i |  |  |  |  |  |
| 3800W6 |  | 6 | 13.5i | 10 i |  |  |  |  |  |
| 3800W8 |  | 8 | 13.5i | 10 i |  |  |  |  |  |
| 3800W12 | 3800 | 12 | 13.5i | 10 i | 1i | 0 | 0 | 6i |  |
| 3800N6S |  | 6 | 11 i | 7.5i |  |  |  |  |  |
| 3800N8S |  | 8 | 11 i | 7.5i |  |  |  |  |  |
| 3800N12S |  | 12 | 11 i | 7.5i |  |  |  |  |  |
| 3800W6S |  | 6 | 13.5i | 7.5i |  |  |  |  |  |
| 3800W8S |  | 8 | 13.5i | 7.5i |  |  |  |  |  |
| 3800W12S |  | 12 | 13.5i | 7.5i |  |  |  |  |  |
| 38PPN |  |  | 8.5i | 10 i |  |  |  |  |  |
| 38PPW |  |  | 13.5i | 10 i | 1i | 0 | .125i | 6i |  |
| 38PPNS |  |  | 11 i | 7.5i |  |  |  |  |  |
| 38PPWS | 3800-3 | (3) | 13.5i | 7.5i |  |  |  |  |  |
| 38PPW90 |  |  | 10 i | 13.5i | .5i | .5i | .5i | 6i |  |
| 38PPNS90 |  |  | 7.5i | 11 i |  |  |  |  |  |
| 38PPW270 |  |  | 10 i | 13.5i |  |  |  |  | page |
| 3820A |  |  | 8.5i | 11 i |  |  |  |  | devices |
| 3820A90 |  |  | 11 i | 8.5i |  |  |  |  |  |
| 3820A180 |  |  | 8.5i | 11 i |  |  |  |  |  |
| 3820A270 | 3820 | (3) | 11 i | 8.5i | 1i | .5i | .5i | 6i |  |
| 3820L |  |  | 8.5i | 14 i |  |  |  |  |  |
| 3820A4 |  |  | 210mm | 297mm |  |  |  |  |  |
| 3820B4 |  |  | 257mm | 364mm |  |  |  |  |  |
| 3820B5 |  |  | 182mm | 257mm |  |  |  |  |  |
| 4250A |  |  | 8.5i | 11 i |  |  |  |  |  |
| 4250B | 4250 | (3) | 11 i | 17 i |  |  |  |  |  |
| 4250L |  |  | 8.5i | 14 i | 1i | .5i | .5i | 6i |  |
| 4250A3 |  |  | 297mm | 420mm |  |  |  |  |  |
| 4250A4 |  |  | 210mm | 297mm |  |  |  |  |  |

1 The physical device type corresponding to the TERM logical device can be either 2741 or 3270, depending upon the actual terminal type.
2 This is a 12-pitch device; all other 1403 devices are 10-pitch.
3 The line spacing for page devices is determined by the .LS [Line Spacing] control word and the fonts used in the document.

**Figure 8. SCRIPT/VS Logical Devices:** This table lists the logical devices that can be specified with the DEVICE option of the SCRIPT command, and the default page dimensions for each.

**CONTINUE:**

Normally, SCRIPT/VS stops processing when it finds an error. This option directs it to continue processing unless the error is a severe one.

**MESSAGE(DELAY ID TRACE):**

This option, with the DELAY parameter specified, causes error messages to be printed at the end of the output document, instead of being displayed at a terminal during processing.[30]

The ID parameter causes SCRIPT/VS to include the error message identifier along with the error message.

The TRACE parameter provides additional information pointing to where the error occurred by tracing through all the imbedded files from the one where the error occurred back to the primary input file.

## *Font Requirements for Page Devices*

SCRIPT/VS has as the original default font Monotype Times New Roman (5771-AAR)[31] for the IBM 4250 printer and Sonoran Serif (5771-ABA)[32] for the IBM 3820 Page Printer and IBM 3800 Printing Subsystem Model 3 and Model 6. These default fonts may have been changed at your installation. If you wish to override the default typeface, specify a different font in the CHARS option of the SCRIPT command. Although you can specify as many fonts on CHARS as you wish, the starter set will use only the first one specified.

The initial font, specified either by the default font or on CHARS, must be a typographic font available in a variety of sizes and weights. Specifying a typewriter style font on CHARS will cause invalid font error messages from SCRIPT/VS.

In addition to requiring a typographic font as the initial (body) font, the starter set also requires the Typewriter and Pi Specials (5771-AAW) font for the IBM 4250 printer and the Pi and Specials (5771-ABC) font for the IBM 3820 Page Printer and IBM 3800 Printing Subsystem Model 3 and Model 6 for use in formatting unordered list bullets.

---

[30] ATMS-III users should refer to the *ATMS-III: Terminal Operator's Guide* for additional information on using SCRIPT VS in your environment, especially when you use the LIB option of the SCRIPT command or when you want your output.

[31] Trademarks of The Monotype Corporation, Limited.

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

# Chapter 13. Common Problems—Symptoms and Solutions

This symptom index does not include those things we think you'll be able to figure out for yourself. For example, if a huge chunk of the document comes out highlighted, it may be traumatic, but you can still figure out for yourself that you left out (or misspelled) an end tag for a highlighted phrase.

This index is for things that are a little more subtle. (It may comfort you a little when you have one of these problems to know that everything we tell you about here is something that has happened to us, and that's how we know about them.)

## Symptom: Unexpected Formatting Results

***If you use SCRIPT/VS control words within your GML documents, you must be careful that they do not counteract the SCRIPT/VS control words that make the tags work as they do.***

For example, if you set indention within a list, the indention would interfere with the indention already set by DSMPROF3 for lists. (See the section entitled "Control Words and Macros" in *DCF: Generalized Markup Language Starter Set Reference* for a list of SCRIPT/VS control words that can be used safely in GML documents.)

So, if you have unexpected results in your formatting, look for any control words (other than those in the list) you may have added to your source file and remove them.

## Symptom: Text Disappears

If pieces of text disappear from your output document, the most likely reason is that you have left out a markup-content separator (the period at the end of the markup).

Many tags don't need the period, but some of them do. If a tag does need an ending period and you have left it out, then any text following that tag will be treated as an attribute and will disappear from your document. What's even worse, if you leave out the period, some things that come out okay today may not work in the future, due to changes in the profile you use. This is why it is a good idea to get into the habit of *always* putting in the period at the end of the markup.

If a lot of text disappears, you may be missing a PSC end tag.

Another thing that can cause text to disappear is an attribute value that contains blanks or special characters and is not enclosed in single quotation marks. The rest of the attribute value after the blank or special character doesn't get picked up as part of that attribute value.

## Symptom: Text Doesn't Format

**Throughout the Document:**

If nothing worked, and your text and tags all come out formatted in huge blocks, then your SCRIPT command did not specify the profile properly and the tags weren't recognized at all.

If your tags don't show up in the text, but it still formats as one big lump, then your SCRIPT command did not specify the library properly. In this case, the tags were recognized, but SCRIPT/VS couldn't find the processing instructions to do the work.

This won't happen to you when you use DSMPROF3, since it checks to make sure the right library is there; if it isn't, it gives you a message and stops processing. This way, you don't waste a processing run that is sure to give you bad results. (We mention it here because it could happen if you are using a profile other than DSMPROF3.)

**Starting in the Middle of the Document:**

When all of a sudden your formatting stops, and the text comes out in big blocks, the likelihood is that you have an extra XMP or FIG end tag in your input document.

You will have to find it and get rid of it.

## Symptom: Column Is Narrow

If you have this problem, it is almost certain that somewhere you have forgotten an end tag—usually for a list.

The starter set does a bit of "housekeeping" for head tags, and if it finds that a list, an example, or a figure is still going on, it ends it and gives you a message. However, if there are no more headings or sections in your document, this check can't be done for you.

It is a good practice to check the last couple of pages of your document and make sure that the column hasn't shrunk; if it has, then you have this problem. Look for an end tag near where the problem begins. If the end tag is missing, put one in.

## Symptom: The Colon after a Level 5 or 6 Heading Is Missing

If you don't get a colon (:) after a level 5 or 6 heading, it's because you haven't included the P tag for the first paragraph following the heading. It is the P tag that tells the starter set to put in the colon. If your heading text is too long or if other tags follow the heading tag, the colon may end up in an unpredictable location.

## Symptom: Page Number References Are Wrong

Sometimes the page numbers in the table of contents, in the list of illustrations, in the list of tables, or in cross-references don't correspond to the pages where the headings, figures, or tables actually appear. If you are using two formatting passes, it is likely that the output pages from the first SCRIPT/VS pass through the document are numbered differently than the output pages from the second pass. Because the table of contents, list of illustrations, and list of tables are built on the first pass and then printed in your document at the beginning of the last pass, they have the page numbers from the first pass rather than the last pass.

The mismatch happens when things such as cross-references to headings are used before they are defined. For example, if you specify two formatting passes and you use a

HDREF tag to refer to a heading that comes *later* in the book, the starter set doesn't know how much space to leave for that heading reference when it encounters the HDREF tag. When the starter set processes the second pass, it then knows how much space to leave for that heading reference and can place it correctly. A heading reference that is shorter or longer than the space left for it can cause the page numbers to be different from one pass to the next.

A quick fix for this problem is to put the TOC, FIGLIST, and TLIST tags at the end of the document (just before the GDOC end tag). This way, these elements will be assigned the page numbers that were generated on the last pass rather than the first, and they will be correct. However, if you put these elements in the back matter, they will print in two-column format.

Another approach to this problem is to use FPASSES to specify more than two formatting passes. (See "FPASSES(n)" on page 121.)

## *Symptom: Figures Are Out of Order*

When a number of figures with different PLACE and WIDTH attributes get bunched together, they are sometimes printed out of order. This is because the figure numbers are assigned in the sequence in which the figures are found in the source file, but other things might cause the figures to be printed in a different sequence. For example, if you specify a figure with a PLACE attribute of TOP, and then follow it with a figure with a PLACE attribute of INLINE, the first figure can be floated past the second figure and thus be out of order.

The fix for this problem depends on your document. You may want to move some figures around, or specify different PLACE attribute values on them.

For this reason, it is a good idea to keep your figures in separate files. This makes them easy to move when you want to move them.

## *Symptom: Blanks Disappear*

Sometimes, in an example or a figure, you have text followed by a string of blanks, which are followed by more text. If the last thing in the first section of text was a tag (for example, to turn off highlighting), SCRIPT/VS may take out your blanks and move the last section of text up against the first.

You can fix this problem by replacing the first blank in the string of blanks with the &rbl. (required blank) symbol.

## Symptom: Semicolon Won't Print

If a semicolon in your source document doesn't print out and you get a line break instead, it's because SCRIPT/VS has misinterpreted this semicolon as a "control word separator."

Use the &semi. symbol in place of the semicolon that won't print and your problem is solved.

| ## Symptom: Front Matter Lists Are Empty

| If the table of contents, list of illustrations, or list of tables are empty, you've forgotten to specify FPASSES or TWOPASS in your SCRIPT command options.

## Symptom: Index Is Empty

If you get a heading for your index but no entries, you've forgotten the INDEX option on the SCRIPT command.

Of course, you won't get an index if you forget your INDEX tag either, because without this tag there is no way for the starter set to know that you want an index and it will therefore make no place for it in your document.

# Chapter 14. Starter Set and SCRIPT/VS Messages

You can expect to get two kinds of error messages:

1. Messages from the starter set begin with + + + and tell you about markup errors that it has detected. These messages are self-explanatory, and tell you on what page the problem occurred. They are listed in the *Document Composition Facility: Generalized Markup Language Starter Set Reference.*

2. Messages from SCRIPT/VS itself, which fall into two categories:

    a. Messages resulting from markup errors that the starter set didn't detect. Since the starter set cannot check for all possible markup errors, some slip past it. However, they can cause problems when SCRIPT/VS is formatting the document, so SCRIPT/VS gives you a message. With the message (if you have the TRACE option in effect on the formatting command) will come an indication of the line number in the source file where the problem was found. That's the place to look to see if your markup is okay.

    b. Messages resulting from an error in the code that interprets your markup.

       Unfortunately, there is no immediately apparent way to tell this kind of message from the kind described above.

       When you have checked your markup and are sure that the error is not a result of something you've done wrong, take these messages *and* a copy of your source file to the document administrator, or whoever in your organization is responsible for trouble-shooting problems in formatting documents.

    These messages may be confusing, because they generally involve an error in the use of a SCRIPT/VS control word. Since your markup consists of tags and not control words, you may wonder what's going on.

    The secret is that all of the formatting work is done with control words, which you don't see (unless something goes wrong). In most cases, what SCRIPT/VS actually does (a process called *interpreting the GML*) when it finds a tag is check the profile (DSMPROF3) to see what is wanted and then go to another file, called a "macro library" and get the set of control words that do the work for that tag. This set of control words is called an *application processing function*, or *APF* for short.

    It is because of this process that it is easy for an organization to change the way things are formatted without having to change the markup in the source files. All that has to be done is change the profile to point to a different APF, or, alternatively, use a different profile that points to a different APF.

There might be times when your output is wrong and you get no messages at all. That is, SCRIPT/VS has no way of knowing that you aren't getting what you want; like all programs, it just does as it's told.

In this case, you should check our symptom index first, to see if the problem is one we already know how to solve ("Chapter 13. Common Problems—Symptoms and Solutions" on page 127). If that doesn't answer it, and you've checked your markup and are sure that the markup isn't wrong, then it is again time to go to your trouble-shooter with the problem.

# Chapter 15. The Cross-Reference Listing

As we mentioned in "Chapter 8. Overall Document Structure" on page 77, DSMPROF3 produces a cross-reference listing at the end of your document unless you ask it not to at run time (SYSVAR X, as described in "Instructions to the Starter Set (SYSVAR)" on page 116). Normally, you would tear it off and throw it away before you published a book, but we have left it here (following the index) for you to examine.

The cross-reference listing contains seven sections: headings, figures, footnotes, list items, tables, index entries, and an imbed trace.

The first six listings are similar, and they show, for every case of a heading, figure, footnote, list item, table, or index entry that was entered with an ID attribute:

1. The name used on the ID attribute
2. The source file that contained it
3. The output page on which it appears
4. The output page(s) on which any references to it appear.

This assists you in checking for any markup errors in your cross-references.

If you'll recall, we had some unmatched REFID attributes in our chapter on cross-references; you'll see these listed in our cross-reference listing with question marks.

## *The Imbed Trace*

The imbed trace shows the files that were used in the formatting of your document and, through indention, the level at which the file is imbedded. (See "Combining Input Files" on page 84 for information on imbedding files.)

The page number shown in the imbed trace is the page that SCRIPT/VS was processing when it encountered the .IM control word. This may be different from the page on which the output of that file occurs; for example, if the first thing in the imbedded file is an H1 tag, DSMPROF3 will skip to the next page before it starts to print the contents of the imbedded file.

The primary input file (the document you named on your SCRIPT command) is not shown in the imbed trace.

If you look at the imbed trace for this book, you'll see that a number of files are used more than once. For example, SS$MODEL is used eight times. "Chapter 18. Summary by Document Element" on page 143 is made up of nine figures with the same general structure. To ensure that the figures were consistent in their structure (and to save work for the writer of this book), the markup that was common to all of the figures was placed in a single file and then reused for each figure.

We point this out to you because it can prove to be a real time-saver for any documents that have repetitive structures.

# *Part Three: Tag Reference*

This part includes:

# Chapter 16. General Rules for Markup Entry

Here are some helpful rules for markup entry, based on the experience of many users of the GML starter set.

- Document elements are identified with GML tags. The tags begin with the GML delimiter, which in the starter set is the colon (:). End tags use the GML end tag delimiter, which in the starter set is the two characters ":e" (colon followed by a lowercase letter e).
- End the markup with a period.
- Start all input lines at the left margin.
- Begin a new input line for all tags except inline quotes, title references, cross-references, and highlighted phrases.
- When you end a sentence, do not enter any more text on that line.
- Do not end input lines with a hyphen.
- When using labeled attributes, enclose values that include blanks or special characters in single quotation marks. For example:

  ```
  :gdoc sec='ABC Company Confidential'.
  ```

  If the value itself includes a single quotation mark, then you enter two of them so that SCRIPT/VS does not think the first one ends the value, like this:

  ```
  :title stitle='Mrs. O''Leary''s Cow'.
  ```

  The actual value that will then be used by SCRIPT/VS is

  ```
  Mrs. O'Leary's Cow
  ```

- When you enter *decimal values* on attributes, enclose the value within a set of single quotation marks. Otherwise, the starter set will interpret the period as a markup *end tag delimiter*, not as a decimal point, and therefore the attribute value will not be processed correctly.

  If you'd rather, you can enter such decimal values using a *comma* to mean a decimal point, and the starter set will then process the attribute appropriately.

- Do not start text lines with a period. SCRIPT/VS will assume that the line is either a control word or a macro, and will give you an error message if it is not (and will not print the line).
- Maximum input line length is 256 characters after all symbols are substituted.

The following practices are also recommended:

- Do not mark an element with a tag that does not correctly describe it, even if it apparently results in the correct processing. The document administrator or text programmer may someday change the way that particular tag formats, and you may find that your document is formatted in a way you did not want.
- Enter tags and attribute labels in lowercase letters.
- Do not enter blank lines.

  If you need a blank line in a figure or an example, just put an &rbl. symbol on the line you want to be blank.
- When storing parts of a document in separate files, be sure each file starts at the beginning of a paragraph unit or higher level element and finishes at the end of one.

# Chapter 17. Alphabetic Summary of Tags

The table which begins on the next page is an alphabetic summary of GML tags. Shown are: each tag, its meaning, its attributes (if any), the allowable text contents, and the end tag (if any).

***Required attributes are in italics.*** The other attributes are optional.

In the contents column, you will find one of these entries:

**Text on same or next line; no other tags.**

The text must be entered either all on the same line as the end of the markup or all on the next line; no tags, such as those for highlighting or quotes, can be used within the text.

The tag should appear at the beginning of the input line.

**No immediate text.**

There is no text directly associated with this tag; that is, the tag is always followed by another tag.

The tag should appear at the beginning of the input line.

**Implied paragraph.**

The text associated with this tag is an implied paragraph. The text can start on the same line as the tag or on the next line and continue for as many lines as needed.

The tag should appear at the beginning of the input line.

**Any text; anywhere on line.**

These tags can appear anywhere within the input line, and the text can be anything that is reasonable in the context of that tag. (You wouldn't, for example, put a figure in the middle of a title reference.)

**Self-contained.**

These tags can appear anywhere within the input line, and the content of the tag is entirely contained within the tag (plus attributes) itself; that is, it has no effect on the following text.

The tags are shown with the starter set default delimiters. (Your organization may use different delimiters.)

| Tag | Element | Attributes | Content | Termination |
|---|---|---|---|---|
| :ABSTRACT | abstract | | No immediate text. | |
| :ADDRESS | address | | No immediate text. | :EADDRESS |
| :ALINE | address line | | Text on same line or next line; no other tags. | |
| :APPENDIX | appendix section | | No immediate text. | |
| :AUTHOR | author of document | | Text on same line or next line; no other tags. | |
| :BACKM | back matter | | No immediate text. | |
| :BODY | body of document | | No immediate text. | |
| :C | cell of a table | (cell number) | Implied paragraph. | :EC |
| :CIT | title citation | | Any text; anywhere on line. | :ECIT |
| :DATE | date | | If any text, must be on same line or next with no other tags. | |
| :DD | definition description | | Implied paragraph. | |
| :DDHD | definition description heading | | Text on same line or next line; no other tags. | |
| :DL | definition list | BREAK<br>COMPACT<br>HEADHI=<br>TERMHI=<br>TSIZE= | No immediate text. | :EDL |
| :DOCNUM | document number | | Text on same line or next line; no other tags. | |
| :DT | definition term | | Text on same line or next line; no other tags. | |
| :DTHD | definition term heading | | Text on same line or next line; no other tags. | |
| :FIG | figure | DEPTH=<br>FRAME=<br>ID=<br>PLACE=<br>WIDTH= | No immediate text. | :EFIG |
| :FIGCAP | figure caption | | Text on same line or next line; no other tags. | |
| :FIGDESC | figure description | | Implied paragraph. | |
| :FIGLIST | figure list | | No immediate text. | |
| :FIGREF | figure reference | PAGE=<br>*REFID=* | Self-contained. | |
| :FN | footnote | ID= | Implied paragraph. | :EFN |
| :FNREF | footnote reference | *REFID=* | Self-contained. | |
| :FRONTM | front matter | | No immediate text. | |
| :GD | glossary definition | | Implied paragraph. | |
| :GDOC | general document | SEC= | No immediate text. | :EGDOC |
| :GL | glossary list | COMPACT<br>TERMHI= | No immediate text. | :EGL |

| Tag | Element | Attributes | Content | Termination |
|---|---|---|---|---|
| :GT | glossary term | | Text on same line or next line; no other tags. | |
| :HDREF | head reference | PAGE= *REFID=* | Self-contained. | |
| :HP0-:HP3 | highlighted phrases 0-3 | | Any text; anywhere on line. | :EHP0- :EHP3 |
| :H0-:H1 | head levels 0-1 | ID= STITLE= | Text on same line or next line; no other tags. | |
| :H2-:H6 | head levels 2-6 | ID= | Text on same line or next line; no other tags. | |
| :IH1-:IH3 | index entry heading levels 1-3 | ID= PRINT= SEE= SEEID= | Text on same line or next line; no other tags. | |
| :INDEX | index | | No immediate text. | |
| :IREF | index entry reference | *REFID=* PG= SEE= SEEID= | Self-contained. | |
| :I1 | index entry term level 1 | ID= PG= | Text on same line or next line; no other tags. | |
| :I2-:I3 | index entry term levels 2-3 | ID= REFID= PG= | Text on same line or next line; no other tags. | |
| :LI | list item | ID= | Implied paragraph. | |
| :LIREF | list item reference | PAGE= *REFID=* | Self-contained. | |
| :LP | list part | | Implied paragraph. | |
| :LQ | long quote (excerpt) | | No immediate text. | :ELQ |
| :NOTE | note | | Implied paragraph. | |
| :OL | ordered list | COMPACT | No immediate text. | :EOL |
| :P | paragraph | | Paragraph. | |
| :PC | paragraph continuation | | Implied paragraph. | |
| :PREFACE | preface | | No immediate text. | |
| :PSC | process specific controls | PROC= | No immediate text. | :EPSC |
| :Q | quote | | Any text; anywhere on line. | :EQ |
| :RDEF | row definition | *ID=* HP= ALIGN= CONCAT= VALIGN= ROTATE= MINDEPTH= ARRANGE= CWIDTHS= | No immediate text. | |

| Tag | Element | Attributes | Content | Termination |
|---|---|---|---|---|
| :ROW | row | REFID=<br>SPLIT= | No immediate text. | :EROW |
| :SL | simple list | COMPACT | No immediate text. | :ESL |
| :TABLE | table | *REFID*=<br>ID=<br>COLUMN<br>PAGE<br>SPLIT=<br>ROTATE=<br>WIDTH= | No immediate text. | :ETABLE |
| :TCAP | table caption | | Text on same line or next line; no other tags. | |
| :TDESC | table description | | Implied paragraph. | |
| :TFT | table footing | REFID= | No immediate text. | :ETFT |
| :THD | table heading | REFID= | No immediate text. | :ETHD |
| :TITLE | title of document | STITLE= | Text on same line or next line; no other tags. | |
| :TITLEP | title page | | No immediate text. | :ETITLEP |
| :TLIST | table list | | No immediate text. | |
| :TNOTE | table notes | | Implied paragraph. | :ETNOTE |
| :TOC | table of contents | | No immediate text. | |
| :TREF | table cross-reference locator | *REFID*=<br>PAGE= | Self-contained. | |
| :UL | unordered list | COMPACT | No immediate text. | :EUL |
| :XMP | example | DEPTH= | No immediate text. | :EXMP |

**Table 6. Alphabetic Summary of Starter Set Tags:** (This alphabetic summary of the starter set tags was entered using the starter set table tags.)

# Chapter 18. Summary by Document Element

| *Major Document Elements* | **Document Type** | general document | :GDOC SEC= :EGDOC |
|---|---|---|---|
| | **Front Matter** | front matter | :FRONTM |
| | | title page | :TITLEP :ETITLEP |
| | | title | :TITLE STITLE= |
| | | document number | :DOCNUM |
| | | date | :DATE |
| | | author | :AUTHOR |
| | | address | :ADDRESS :EADDRESS |
| | | address line | :ALINE |
| | | abstract | :ABSTRACT |
| | | preface | :PREFACE |
| | | table of contents | :TOC |
| | | figure list | :FIGLIST |
| | | table list | :TLIST |
| | **Body** | body | :BODY |
| | **Appendixes** | appendix section | :APPENDIX |
| | **Back Matter** | back matter | :BACKM |

| *Headings* | head level 0-1 | :H0-:H1 ID= STITLE= |
|---|---|---|
| | head level 2-6 | :H2-:H6 ID= |
| | head reference | :HDREF PAGE= REFID= |
| | table of contents | :TOC |

| *Basic Text* | paragraph | :P |
|---|---|---|
| | paragraph continuation | :PC |
| | highlighted phrase 0-3 | :HP0-:HP3<br>:EHP0-:EHP3 |
| | title citation | :CIT<br>:ECIT |
| | address | :ADDRESS<br>:EADDRESS |
| | address line | :ALINE |
| | note | :NOTE |
| | quotation (inline) | :Q<br>:EQ |
| | quotation (excerpt) | :LQ<br>:ELQ |

| *Examples and Figures* | **Examples** | example | :XMP<br>DEPTH=<br>:EXMP |
|---|---|---|---|
| | **Figures** | figure | :FIG<br>DEPTH=<br>FRAME=<br>ID=<br>PLACE=<br>WIDTH=<br>:EFIG |
| | | figure caption | :FIGCAP |
| | | figure description | :FIGDESC |
| | | figure reference | :FIGREF<br>PAGE=<br>REFID= |
| | | figure list | :FIGLIST |

| *Indexing* | index | :INDEX |
|---|---|---|
| | index entry term (levels 1-3) | :I1-:I3<br>ID=<br>PG=<br>REFID= |
| | index entry heading (levels 1-3) | :IH1-:IH3<br>ID=<br>PRINT=<br>SEE=<br>SEEID= |
| | index entry reference | :IREF<br>REFID=<br>PG=<br>SEE=<br>SEEID= |

| | | |
|---|---|---|
| ***Tables*** | defining the table | :RDEF<br>ID=<br>HP=<br>ALIGN=<br>CONCAT=<br>VALIGN=<br>ROTATE=<br>MINDEPTH=<br>ARRANGE=<br>CWIDTHS= |
| | building the table | :TABLE<br>REFID=<br>ID=<br>COLUMN<br>PAGE<br>SPLIT=<br>ROTATE=<br>WIDTH=<br>:ETABLE |
| | row | :ROW<br>REFID=<br>SPLIT=<br>:EROW |
| | cell | :C<br>(cell number)<br>:EC |
| | table heading | :THD<br>REFID=<br>:ETHD |
| | table footing | :TFT<br>REFID=<br>:ETFT |
| | table reference | :TREF<br>REFID=<br>PAGE= |
| | table caption | :TCAP |
| | table description | :TDESC |
| | table notes | :TNOTE<br>:ETNOTE |
| | table list | :TLIST |

| | | |
|---|---|---|
| ***Footnotes*** | footnote | :FN<br>ID=<br>:EFN |
| | footnote reference | :FNREF<br>REFID= |

| | | | |
|---|---|---|---|
| *Lists* | **Simple** | simple list | :SL<br>COMPACT<br>:ESL |
| | | list item | :LI<br>ID= |
| | **Unordered** | unordered list | :UL<br>COMPACT<br>:EUL |
| | | list item | :LI<br>ID= |
| | **Ordered** | ordered list | :OL<br>COMPACT<br>:EOL |
| | | list item | :LI<br>ID= |
| | **Definition** | definition list | :DL<br>COMPACT<br>BREAK<br>HEADHI=<br>TERMHI=<br>TSIZE=<br>:EDL |
| | | definition term heading | :DTHD |
| | | definition description heading | :DDHD |
| | | definition term | :DT |
| | | definition description | :DD |
| | **Glossary** | glossary list | :GL<br>COMPACT<br>TERMHI=<br>:EGL |
| | | glossary term | :GT |
| | | glossary description | :GD |
| | **List Part** | list part | :LP |
| | **List Item Reference** | list item reference | :LIREF<br>REFID= |

| | | |
|---|---|---|
| *Process-Specific* | process-specific controls | :PSC<br>PROC=<br>:EPSC |

# *Appendixes*

This part contains the following appendixes:

# Appendix A. Creating and Editing a File

This chapter contains a discussion of how to use an editor and 3277 display terminal to create and edit (make changes to) documents in several common operating systems. The intent of this appendix is to get you started in creating and editing a document. **You will need to refer to specific edit publications for complete information on how to use each editor.** An editor allows you to:

- Create documents and give that document a name.
- Edit documents to revise or make changes to the text that already exists in your document. For example, you can change text by:
  - Writing over it
  - Inserting words or characters within or between existing words
  - Adding lines of new text between lines of existing text
  - Deleting characters, words or entire lines
  - Moving lines or blocks of text around in a document
  - Copying lines or blocks of text from one place in a document to another place in the document.
- File your document—put it into storage until you want to look at it again.
- Return to your document by requesting that document by the name given to the document when it was created.

## *Before You Begin*

Before you can begin to create documents, you need to find out which operating system your installation uses and the editors available.

If you don't already know which operating system and editor you should be using, check with your supervisor or the document administrator at your location. You need to know how to establish communication with the system your organization uses.

Once you've established that communication, you can create your document using an editor. There are many different editors. They all have the same purpose: they allow you to create and manipulate the text in your documents.

The following pages tell you how to create documents using one of the available editors for each of the following systems:

| | |
|---|---|
| **VM/CMS** | See "Creating Documents Using VM/CMS" on page 150 |
| **MVS/TSO** | See "Creating Documents Using MVS/TSO" on page 159 |
| **CICS/ATMS-III** | See "Creating Documents in ATMS-III/CICS" on page 169. |

## *Creating Documents Using VM/CMS*

In this section, we'll briefly discuss a way to use the VM System Product Editor (XEDIT) to create and edit documents on the VM operating system using a 3277 display terminal. For complete information on using XEDIT, see *IBM Virtual Machine/System Product: System Product Editor User's Guide*, SC24-5220.

Once you have established communication with your computer and you see an

```
R;
```

on your screen, you are ready to prepare a document.

### Creating and Naming a Document

With XEDIT, you really create and name a document (or access a document that already exists) at the same time that you invoke (call into operation) the editor.

The following subcommand

```
XEDIT MYDOC SCRIPT
```

entered on the command line at your terminal creates a document that is named "mydoc script."

```
mydoc is the filename
script is the filetype
```

The filename and filetype can each be up to eight characters long.

**Note:** In all the examples given below, the subcommands can be typed in with either upper- or lowercase characters. We have entered the subcommands in uppercase strictly for purposes of illustration. Press the ENTER key after the subcommands have been typed in to communicate them to the computer.

### Entering Text

Once you have entered the subcommand,

```
XEDIT MYDOC SCRIPT
```

the document with that name appears on your screen and you are in *edit mode* and ready to enter text or edit (make changes to) your document.

The top of the document you created or accessed will be displayed on your terminal. On your screen you will find an arrow with the cursor beside it. The arrow looks like this:

```
====>
```

This arrow designates the XEDIT subcommand line and appears on every screen as you use XEDIT.

The cursor (the little line that looks like this _) shows where any typing you enter will appear. The cursor can be moved up, down, left, or right on the screen by using the four keys showing the arrows on the right-hand side of the keyboard. Pressing the ENTER key will move the cursor to the command line. Depending on what you are doing at the time, you may have to press the ENTER key twice to get the cursor to the command line.

Now that you are in *edit mode*, one of the things you will probably want to do is enter text into your file. You can enter text using either of the following methods:

- Edit mode
- Input mode

**Edit Mode:** Edit mode is ordinarily used when you want to enter small segments of text—words, phrases, one or two lines. For more information on adding text in the edit mode, see "Adding Text" on page 153.

**Input Mode:** Input mode is ordinarily used when you want to enter a lot of text.

To use the input mode, enter:

```
INPUT
```

on the command line. The entire screen becomes available to enter text. It's a good idea to begin each sentence on a new line and to keep the lines fairly short. When you want to start a new line of text while in input mode, you must do the equivalent of a carriage return by pressing the *tab key* (it's the key with an arrow on it on the left-hand side of your keyboard).

When you run out of space on the screen and want more, simply press the ENTER key once. This pushes the bottom of the document to the top of the screen and provides you with room to continue entering text. When you want to stop using input mode, press the ENTER key twice. That puts you back into edit mode.

Anytime that you are entering text onto the screen, whether you are in input mode or edit mode, you can type over any words you want to change. Simply place the cursor under the first of the group of letters you want to change and type right over them. If you want to insert text between letters, press the *insert mode* key that appears on the right-hand side of the keyboard. (It says:

```
INS
MODE
```

on the key.) Place the cursor under the letter that you want the inserted text to appear before. Type in the text you want inserted and the text already there will shift to the right. (If the text already there will not shift to the right, position the cursor on the XEDIT subcommand line and enter

```
NULLS ON
```

to allow the shifting of text to take place.) When you have finished inserting text, press the *reset key* that appears in the lower left-hand corner of the keyboard. This ends insert mode.

If you need to delete (remove) any characters from a line of text, position the cursor under the first letter of the portion you want deleted and press the *delete key* once for each character you want deleted. The delete key is positioned next to the insert key. (It says:

```
DEL
```

on the key.)

## Editing Text

When you are working with a document, you will need to edit your text frequently. You will need to understand the concept of the current line and the prefix area so that you can add, delete, move, copy, and change text within your document.

### *Current Line*

The current line is the line on your screen that is brighter and bolder than the rest of the lines on the screen display. You can change which line is the current line with any of the following methods.

- On the command line, enter:

  ```
  TOP
  ```

  and the top line of the document becomes the current line

- On the command line, enter:

  ```
  BOTTOM
  ```

  and the bottom line of the document becomes the current line

- On the command line, enter UPn (where n is any number of lines); for example:

  ```
  UP10
  ```

  and the current line becomes the line specified (in this example by moving toward the top of the file 10 lines)

- On the command line, enter DOWNn (where n is any number of lines); for example:

  ```
  DOWN6
  ```

  and the current line becomes the line specified (in this example by moving toward the bottom of the document 6 lines)

- On the command line, enter :n (where n is a specific line number); for example:

  ```
  :56
  ```

  and the current line (in this example) becomes line 56 of your document.

### *Prefix Area*

The first *or* last five characters of each line are called the prefix area. If line numbering is on, the prefix area contains the line number of each line. If line numbering is off, the prefix area looks like this:

```
=====
```

You cannot make any entries into the prefix area if you are in *insert mode*. Press the *reset key* if you *lock* the keyboard by trying to make entries in the prefix area while in insert mode. If the keyboard locks, you won't be able to move the cursor around on the screen or enter any text.

Like the subcommands, entries into the prefix area can be either upper- or lowercase, and are the easiest and most common way to add, delete, move, and copy text. Enter a / in the prefix area of any line and that line becomes the current line.

## *Adding Text*

You can add blank lines to a document for adding more text in several ways.

- Enter the following on the command line:

  ```
  :N ADDx
  ```

  The **N** is a specific line number in your document after which you want to add the blank lines, the **x** is the number of blank lines you want added. In the following example, the subcommand requests 5 blank lines after line number 10.

  ```
  :10 ADD5
  ```

- Type the single character:

  ```
  A
  ```

  in the first position of the prefix area of any line. This causes one blank line to be added immediately following the line containing the A.

- Type:

  ```
  An (where n is a number)
  ```

  at the beginning of the prefix area of any line. The number of blank lines specified with the **n** will be inserted after the line containing the **An**. For example,

  ```
  A10== will add 10 blank lines after this line.
  A28== will add 28 blank lines after this line.
  ```

## *Deleting Text*

Text can be deleted in the following ways:

- Enter:

  ```
  DELETE
  ```

  on the command line. This deletes the current line only.

- Enter:

  ```
  DELETE *
  ```

  on the command line. This deletes your document beginning with the current line *to the end of the document.*

- Enter:

  ```
  :n DELETEx (where n is a specific line number and
             x is a specified number of lines).
  ```

  The following subcommand, for example, tells the system to delete 13 lines starting with line 22

  ```
  :22 DELETE13
  ```

- Enter:

  ```
  D
  ```

  in the prefix area. This deletes one line; the line on which the **D** is entered.

- Enter:

  ```
  Dn (n is a number)
  ```

  in the prefix area. The number of lines specified by **n** will be deleted beginning with the line on which you enter the **Dn**. In the following example, 10 lines will be deleted starting with the line on which the request appears.

  ```
  D10== The ten lines to be deleted start with this one.
  ```

- Enter:

  ```
  DD
  ```

  in the prefix area of the first line and in the prefix area of the last line of a block of text to be deleted. The block of lines will be deleted. The following example shows how to delete a block of several lines of text.

  ```
  01344 This text will stay.
  DD345 Delete begins here.
  01346 This line gets deleted.
  01347 This line gets deleted.
  DD348 Delete ends here.
  01349 This line will also stay.
  ```

  results in:

  ```
  01344 This text will stay.
  01345 This line will also stay.
  ```

### *Moving Text*

The MOVE operation removes some line(s) from their current location and inserts them in another part of the document. Text can be moved in several different ways:

- Enter:

  ```
  MOVE from to
  ```

  on the command line (where "from" is the number of lines you want moved beginning with the current line and "to" is how many lines down in the document you want the moved lines to be inserted).

  For example, to move 2 lines beginning with the current line and place them after the 5th line down from the current line, you would enter the following:

  ```
  MOVE 2 5
  ```

- Enter the single character:

  ```
  M
  ```

  in the prefix area of any line you want to move. Then enter a **P** for "preceding" or an **F** for "following" in another prefix to tell the system whether the line to be moved precedes or follows the line indicated.

  In the following example, line 2 will be moved. It will be inserted following line 4.

  ```
  00001 This is line 1.
  M0002 This is line 2.
  00003 This is line 3.
  F0004 This is line 4.
  ```

  results in:

  ```
  00001 This is line 1.
  00002 This is line 3.
  00003 This is line 4.
  00004 This is line 2.
  ```

- Enter:

  ```
  Mn (where n is a specified number of lines to be moved)
  ```

  in the prefix of any line. Then indicate the location of the move by entering a **P** or an **F** in the line that the moved lines will precede or follow. In the example below, for instance, lines 1 and 2 will be moved to a location immediately preceding line 4.

  ```
  M2001 This is line 1.
  00002 This is line 2.
  00003 This is line 3.
  P0004 This is line 4.
  ```

  results in:

  ```
  00001 This is line 3.
  00002 This is line 1.
  00003 This is line 2.
  00004 This is line 4.
  ```

- Enter:

  ```
  MM
  ```

  in the prefix areas of the first line and last line of a block of text you want to move. Remember to use a **P** or an **F** to designate the location of the move. In the example below, lines 1 through 3 are moved to a location directly after line 4.

  ```
  MM001 This is line 1.
  00002 This is line 2.
  MM003 This is line 3.
  F0004 This is line 4.
  ```

results in:

```
00001 This is line 4.
00002 This is line 1.
00003 This is line 2.
00004 This is line 3.
```

## *Copying Text*

The COPY operation leaves the original line(s) in place as they were and duplicates the line(s) at the assigned destination. You can copy text in the following ways (note how similar the process is to moving text):

- Enter:

  ```
  COPY from to
  ```

  on the command line (where "from" is the number of lines you want copied beginning with the current line and "to" is how many lines down in the document you want the copied lines to be inserted).

  For example, to copy 2 lines beginning with the current line and place them after the 5th line down from the current line, you would enter the following:

  ```
  COPY 2 5
  ```

- Enter **C**, **Cn** or **CC** in the prefix of a line to copy a single line, several lines or a block of lines, respectively. Be sure to show the destination of the copied lines by putting a **P** or an **F** in the prefix of the destination line.

## *Changing Text*

To change text with a subcommand, you must make sure that the character(s) or line(s) that you wish to change text in appear on the current line or after the current line in your document.

Once you have located the text you want to change, you can change it in several ways.

- **Single changes**—Replacing one character string or word with another. The format for such a change is as follows:

  ```
  CHANGE/oldword/newword/
  ```

  A specific example would be to assume that the following line is the current line:

  ```
  00001 He is what he is.
  ```

  If you issue the following subcommand:

  ```
  CHANGE/is/was/
  ```

  the current line would now look like this:

  ```
  00001 He was what he is.
  ```

Note that only the first occurrence of "is" was changed. To change further occurrences of "is" you could repeat the single change subcommand or issue the following subcommand:

```
change/is/was/1 *
```

where:

1 means make the change requested on one line only

* means to change *every* occurrence of "is" on that one line to "was"

- **Global changes**—Changes that modify every occurrence of a designated word or character string from the current line to the end of a document. To make a global change for a complete document, first make the **TOP** of the document the current line and then enter:

  ```
  CHANGE/oldword/newword/* *
  ```

  on the command line. This will change every occurrence of the designated word or string on every line, as shown in the following example:

  ```
  00000 TOP OF FILE
  00001 flowers is flowers is flowers.
  00001 flowers is flowers is flowers.
  00002 flowers is flowers is flowers.
  00004 * * * END OF FILE * * *
  ```

  If you enter:

  ```
        CHANGE/flowers/a Rose/* *
  ```

  the result is:

  ```
  00000 TOP OF FILE
  00001 a Rose is a Rose is a Rose.
  00002 a Rose is a Rose is a Rose.
  00003 a Rose is a Rose is a Rose.
  00004 * * * END OF FILE * * *
  ```

  (with apologies to Gertrude Stein).

## Saving and Filing Text

- Enter:

  ```
  SAVE
  ```

  on the command line. This makes a permanent record of the document and allows you to continue editing.

- Enter:

  ```
  FILE
  ```

  on the command line. This also makes a permanent record of your document but it takes you out of the editing session.

## Ending the Editing Session

There are several ways to end an editing session.

- Enter:

  ```
  FILE
  ```

  on the command line. This makes a permanent record of the document you were working on and takes you out of the editing session.

- Enter:

  ```
  QUIT
  ```

  on the command line. This allows you to end an editing session if you haven't made any changes in the document you are editing.

- Enter:

  ```
  QQUIT
  ```

  on the command line. This enables you to end an editing session *but it eliminates any changes you may have made to your document during that editing session.*

**Note:** Remember, what we have described in this section is one way to use XEDIT to create and edit a file. There is usually more than one way to achieve your desired results. Your location may have procedures in place that differ from what we have described. See your supervisor or document administrator for complete instructions for creating and editing documents at your location.

## *Creating Documents Using MVS/TSO*

In this section, we'll discuss a way to use ISPF/PDF on the MVS/TSO operating system to create and edit documents using a 3277 display terminal. (The ISPF/PDF editor is also available in CMS. The user commands are identical for both CMS and TSO.) For complete information on using ISPF/PDF in MVS, see *Interactive System Productivity Facility/Program Development Facility for MVS: Program Reference.*

## *Before You Begin*

Before you prepare a document, you'll need to know about the:

- Cursor
- PF Keys
- ENTER Key

The cursor, or little line (_), shows where any typing you enter will appear on the screen. The cursor can be moved up, down, left, or right on the screen by using the four keys showing the arrows located on the right-hand side of the keyboard.

The key with the arrow on the left-hand side of the keyboard (the tab key) advances the cursor from one *field* to the next on the screen, depending on the content of the screen at the time.

There are numbered *Program Functions keys* (PF keys) located on the far right-hand side of the screen. They are used to request commonly used ISPF operations. We'll discuss the PF keys we will be using as we go through our procedures.

You need to press the ENTER key whenever you type in information to transmit that information to the computer. Whenever we say to *enter* something, we mean to type it in and then to press the ENTER key.

Once you have established communication with your computer and you see:

```
READY
```

on your screen, you are ready to prepare a document.

## *Allocating Space and Naming a Document*

When using ISPF in MVS, it's necessary to *allocate* space for a document at the time it's created. (VM will allocate space automatically.)

The following steps explain one way to create a document.

1. Type in ISPF. Press the ENTER key.
2. A **primary option** menu will appear on the screen.
3. On the **Select Option** line, enter:

   ```
   3.2
   ```

4. A **Dataset Utilities** menu will appear on the screen. Type in:

   ```
   A
   ```

   (for "allocate new dataset") after the **Select Option** line. Move the cursor down (using the tab key to move through the fields) to the line that looks like this:

   ```
   DATASET NAME ===>
   ```

   and type in a filename after the arrow, for example:

   ```
   mydoc.text
   ```

   The portion of the name that appears before the

   ```
    .text
   ```

   can be a descriptive term (up to 8 characters long) to identify the document.

   The ".text" follows the descriptive portion of the filename for documents that contain text.

   Now, press the ENTER key.

5. Another menu called **Allocate New Dataset** will appear on the screen.

   The following information will be part of this menu:

   ```
   Volume Serial    ===> (leave blank)
   Space Units      ===> BLKS
   Primary Quan     ===> 1
   Secondary Quan   ===> 1
   Directory Blocks ===> 0
   Record Format    ===> VB
   Record Length    ===> 132
   Block Size       ===> 3120
   ```

   Make the entries following the arrows (= = = >) appear on the screen as shown above.

6. Press the ENTER key and you will return to the previous menu where you specified the allocation.

7. Press the PF3 key and you will return to the **primary option menu**.

You now have a document named "mydoc.text" with space allocated for it.

## Accessing a File

Because you are already in the **primary option menu**, enter:

```
2
```

(for edit) after **Select Option**.

another menu (**Edit/Entry**) will appear.

Position the cursor (using the tab key to move through fields) after

```
DATASET NAME ===>
```

and enter the name of the document you want to use. For example:

```
mydoc.text
```

Because this is a newly created file, the entire screen will become available for text input (except for the top two lines).

- The left-hand side of the top line gives you the name of your document
- The right-hand side of the top line gives you messages and will change as you perform various operations within your document.
- The left-hand side of the second line is the **command input line** where you issue commands to the system
- The right-hand side of the second line tells you the extent of any scroll you do. (This will be discussed more fully under "Scrolling" on page 163.)

It's a good idea to check for the profile settings before you begin entering your text. Enter the word

```
PROFILE
```

on the command input line. The profile settings are displayed on the screen. The ones we are concerned with right now are whether

```
CAPS are OFF
NULLS are ON
RECOVERY is ON
```

CAPS must be OFF to allow your text to appear in both upper- and lowercase.

NULLS must be ON to allow characters to be inserted on the text lines.

RECOVERY must be ON to allow you to recover information entered into a document that would otherwise be lost if there is a system interruption.

If CAPS are not off, position the cursor on the **command input line** (you can always place the cursor immediately at the **command input line** by pressing the PF 12 key) and enter:

```
CAPS OFF
```

If NULLS are not on, position the cursor on the **command input line** and enter:

```
NULLS ON
```

If RECOVERY is not on, position the cursor on the **command input line** and enter:

```
RECOVERY ON
```

Type the word:

```
RESET
```

on the command input line to rid the screen of the display of the profile settings.

You're now ready to begin entering text.

Because we did a reset, the blank lines are gone from the screen. Position the cursor at the very beginning of the line that says

```
****** **************TOP OF DATA*****************
```

Enter an:

```
I 20
```

and a screen of 20 blank lines will appear with:

```
 ......
```

preceding each line.

The six dots are what is called the *line command area.* Move the cursor to the right of the dots (using the tab key) and you can begin entering text. Once you've filled in all the blank lines with text, press the ENTER key while the cursor is still on the last line of text and another blank line will appear. As long as you continue to press the ENTER key at the end of each line of text you add, a new blank line will appear. (You are in *continuous insert mode.*)

The lines of text you have added will be numbered in the line command area each time you press ENTER. (It's a good idea to keep your lines of text fairly short and to begin each new sentence on a new line.)

If you accidentally press ENTER on a blank line, the blank line will disappear. If you need to continue to enter text, simply enter an **I** in the **line command area** of the last line of text and you will again be in continuous insert mode.

Anytime that you are entering text onto the screen, you can type over any words you want to change. Simply place the cursor under the first of the group of letters you want to change and type right over them. If you want to insert text between letters, press the *insert mode* key that appears on the right-hand side of the keyboard. It says:

```
INS
MODE
```

on the key. Place the cursor under the letter that you want the inserted text to appear before. Type in the text you want inserted and the text already there will shift to the right. When you have finished inserting text, press the *reset key* that appears in the lower left-hand corner of the keyboard.

If you need to delete (remove) any characters from a line of text, position the cursor under the first letter of the portion you want deleted and press the *delete key* once for each character you want deleted. The delete key is positioned next to the insert key. It says:

```
DEL
```

on the key.

When you have finished entering as much text as you want, press the ENTER key while you are on the blank line. The blank line will disappear and the cursor will be positioned on the command line.

## Editing Text with Line Commands

When you are working with a document, you will need to edit (make changes to) your text frequently.

To edit your document you will also need to know how to *scroll.*

### *Scrolling*

Scrolling is moving up and down through your file. You can use the PF7 key to scroll up in your document and the PF8 key to scroll down in your document. The second line down on the right-hand side of your screen tells you how much of the screen will scroll each time you push either one of the scroll PF keys.

You can change the amount of the scroll by moving the cursor to the scroll field and typing over the displayed amount. Valid scroll amounts are:

- A number from 1 to 9999, which specifies the number of lines (up or down)
- PAGE - specifies scrolling one page (or screen)
- HALF - specifies scrolling by a half page (or half screen)
- MAX - specifies scrolling to the top or bottom, depending on whether the UP or DOWN PF key is used
- CUR - specifies scrolling based on the where the cursor is. The line on which the cursor is positioned is moved to the top or bottom of the screen, depending on whether the UP or DOWN PF key is used. (If the cursor is not in the body of the screen, or is already positioned at the top or bottom, a full-page scroll will occur.)

Now that you know how to scroll through your document, you are ready to learn some specific and very common editing tasks. These tasks are described below.

### *Adding Text*

Text can be added anywhere in your document by going into continuous insert mode as explained above in "Entering Text".

You can also request a specific number of blank lines, instead of just one at a time by using **In** in the line command area, where **n** is a number; like this:

```
I60005
```

This will add 6 blank lines immediately following the line containing the **I6**.

Move from line to new blank line with the tab key. Once you press ENTER, any remaining blank lines will disappear.

### *Deleting Text*

You can delete lines of text in much the same way that you insert them.
Enter:

```
D
```

in the **line command area** of any line you wish to delete. This deletes one line; the line on which the **D** is entered.

Enter:

```
Dn (where n is a number)
```

in the **line command area**. The number of lines specified by **n** will be deleted beginning with the line on which you enter the **Dn** In the following example, 10 lines will be deleted starting with the line on which the request appears.

```
D10011 The ten lines to be deleted start with this one.
```

Enter:

```
DD
```

on the **line command area** of the first and last lines of a block of lines you wish to delete. (You can scroll to another screen to put in the second, or ending, **DD**.) The following example shows how to delete a block of several lines of text.

```
001234 This text will stay.
DD1235 This is the first line in the block to be deleted.
001236 This line is deleted.
001237 This line is deleted.
DD1238 This is the last line in the block to be deleted.
001239 This line will also stay.
```

results in:

```
001234 This text will stay.
001235 This line will also stay.
```

## *Moving Text*

Moving text is much like deleting or copying text.

Enter the single character:

```
M
```

on the **line command area** of any line you want to move. Then enter *either* a **B** for "before" *or* an **A** for "after" on another **line command area** to tell the system whether the line to be moved comes before or after the line indicated.

In the following example, line 2 will be moved. It will be inserted following line 4.

```
000001 This is line 1.
M00002 This is line 2.
000003 This is line 3.
A00004 This is line 4.
```

results in:

```
000001 This is line 1.
000002 This is line 3.
000003 This is line 4.
000004 This is line 2.
```

Enter:

```
Mn (n is a specified number of lines to be moved)
```

on the **line command area** of any line. Then indicate the location of the move by entering a **B** for "before" or an **A** for "after" on the **line command area** of the line which the moved lines will come before or after. In the example below, for instance, lines 1 and 2 will be moved to a location immediately before line 4.

```
M20001 This is line 1.
000002 This is line 2.
000003 This is line 3.
B00004 This is line 4.
```

results in:

```
000001 This is line 3.
000002 This is line 1.
000003 This is line 2.
000004 This is line 4.
```

Enter:

```
MM
```

on the **line command area** of the first line and last line of a block of text you want to move. Remember to use a **B** or an **A** to designate the location of the move. In the example below, lines 1 through 3 are moved to a location directly after line 4.

```
MM0001 This is the first line in the block that will be moved.
000002 This line will be moved.
MM0003 This is the last line in the block that will be moved.
A00004 The moved lines will follow this line.
```

results in:

```
000001 The moved lines will follow this line.
000002 This is the first line in the block that will be moved.
000003 This line will be moved.
000004 This is the last line in the block that will be moved.
```

Remember that you can scroll to a different screen to continue your MOVE instructions.

## *Copying Text*

The COPY operation leaves the original line(s) in place as they were and duplicates the line(s) at the assigned destination. Notice how similar the process is to moving text.

Enter:

```
C, Cn, or CC
```

on the **line command area** of a line to copy a single line, several lines, or a block of lines, respectively. Be sure to show the destination of the copied lines by putting a **B** or an **A** on the **line command area** of the destination line.

For example:

```
C00001 Leaves this line here and makes a copy of it.
000002 This is line two.
000003 This is line three.
A00004 A copy of line one will appear after this line.
```

results in:

```
000001 Leaves this line here and makes a copy of it.
000002 This is line two.
000003 This is line three.
000004 A copy of line one will appear after this line.
000005 Leaves this line here and makes a copy of it.
```

## *Changing Text from the Command Input Line*

You may want to find a specific portion of text within your document before making a change. You can do this by using the FIND command. When you request that the system find a particular string of characters, the search will begin from the first line that appears on your screen to the end of the document. So, if you want to search through the entire document, be sure to scroll to the top of your document so that the first line of the document is the top line on your screen.

Then you can enter:

```
find 'my word'
```

on the **command input line**.

The system will find the first occurrence of "my word" and place the cursor beneath it.

**Note:** Notice that the text we asked the system to "find" is enclosed in single quotation marks. If there are any blanks in the text you want found, you must put the quotation marks around the text. If you are looking for a single word, the quotation marks are not needed.

If you want to find further occurrences of the string, press the PF5 key. This repeats the last FIND command that you entered and positions the cursor under the next occurrence of the requested string.

Once you have located the text you want to change, you can change it by typing over it or deleting unwanted characters. You can also use the CHANGE command.

- **Single changes**—Replace one character string or word with another. The format for such a change is as follows:

  ```
  CHANGE oldword newword
  ```

  For example, if you had the following line on your screen:

  ```
  000001 He is what he is.
  ```

  You could enter the following CHANGE command on the input command line:

  ```
  CHANGE is was
  ```

The following line is the result of that change request.

```
000001 He was what he is.
```

The first occurrence of the word "is" on the screen was changed. In order to change further occurrences of "is" you would have to repeat the single change command or use one of the methods given below.

- **Selective changes**—Changes that apply only in selected cases.

  Use the same format:

  ```
  CHANGE oldword newword
  ```

  but, rather than pressing the ENTER key, press the PF5 key. Pressing the PF5 key positions the cursor under the first occurrence of the possible change (in this case, "is") If you wish to change the located character string or word, press the PF6 key. If not, press the PF5 key again and the cursor will be positioned under the next occurrence of the character string (oldword) from the CHANGE command. By repeating this process, you can locate all occurrences of a word or string and make any changes to them that you wish.

- **Global changes**—Changes that modify every occurrence of a designated word or character string from the top line on the screen to the end of a document.

  To make a global change for an entire file, make the TOP of the document appear at the top of the screen and then enter:

  ```
  CHANGE oldword newword ALL
  ```

  on the command input line. This will alter every occurrence of the designated word or string as shown in the following example:

  If your document currently looks like this:

  ```
  ****** ************TOP OF FILE*************
  000001 a flower is a flower is a flower.
  000002 a flower is a flower is a flower.
  000003 a flower is a flower is a flower.
  ****** ************BOTTOM OF FILE**********
  ```

  and you enter the following command on the command input line:

  ```
          CHANGE flower Rose ALL
  ```

  your document would then look like this:

  ```
  ****** ************TOP OF FILE*************
  --CHG> A Rose is a Rose is a Rose.
  --CHG> A Rose is a Rose is a Rose.
  --CHG> A Rose is a Rose is a Rose.
  ****** ************BOTTOM OF FILE**********
  ```

Each line on which there was a change made will be identified as being changed in the **line command area**. If you want to rid your screen of the --CHG> message, ENTER "reset" on the **command input line**.

Keep in mind that if the string you want to change contains blanks or commas, you must enclose the string in single quotation marks. If you are changing a single word, the quotation marks are not necessary.

## Saving and Filing Text

Once you have entered new text or changed old text you will want to save it. There are two ways of doing so:

- Enter SAVE on the command input line. This makes a permanent record of the document and allows you to continue editing.
- Press the PF3 key. This also makes a permanent record of your document but it takes you out of the editing session.

## Ending the ISPF Session

Once you have pressed the PF3 key to end an editing session:

1. You will be returned to the **Edit/Entry** panel.
2. Press the PF3 key again. You will be returned to the **Primary Option** menu.
3. Press the PF3 key once more. If you enter the **Specify Disposition of Log Dataset** menu, type in:

   ```
   D
   ```

   on the **Process Option** line and press ENTER. This will take you out of ISPF. You will once again see the word:

   ```
   READY
   ```

   on your screen. You can then log off (disconnect from the TSO system) or go on to another document.

**Note:** Remember, what we have described in this section is one way to use ISPF to create and edit a file. There is usually more than one way to achieve your desired results. Your location may have procedures in place that differ from what we have described. See your supervisor or document administrator for complete instructions for creating and editing documents at your location.

## *Creating Documents in ATMS-III/CICS*

In this section, we'll discuss a way to use ATMS-III to create and edit documents using a 3277 display terminal. This is a brief discussion and not all variations of commands are shown. For complete information on using ATMS-III, see the *Advanced Text Management System-III Terminal Operator's Guide*, SH20-2425. The *Advanced Text Management System-III Quick Reference*, GX20-2354, is a card that lists the syntax of all ATMS commands.

## *Setting Up to Enter/Edit a Document*

Once you have signed on to ATMS-III, you can begin creation of a new document or begin editing a document stored during a previous session. First you will need to check the status of some parameters affecting entry and display of text. If necessary, these may be changed prior to beginning entry or edit.

### Depth Value Setting

If you are going to edit a document previously stored, you can skip this explanation.

Depth is a parameter that affects the numbering of *text units* (strings of characters) and the division of text units into stored *pages*. During entry of text, ATMS-III assigns a unit number to each character string ending with a carrier return character. You enter the carrier return character (see next subsection), or one is inserted between words when the characters entered exceed the LNGTH value displayed at the bottom of your screen. The units are added to a stored page until the current DEPTH value is exceeded. Then, additional text units are added to the next page, and so on. Therefore, each character string entered is given a two-part number referred to as page and unit: 1.1, 1.2, 2.1, and so on. These numbers are used during editing to locate and change the text within a document.

The size of a stored page affects editing efficiency. It is normally more efficient to edit documents when stored pages fit within a single screen. Stored pages normally will fit on a single screen if you set DEPTH to a value less than screen size. For instance, with a display of 24 lines, the DEPTH might be set to 20 or less.

The command to list current depth (along with width and tab values) is:

```
?
```

The command to set depth is:

```
w;;n
```

where n = number of units from 5 to 99.

The setting for depth for looking at a document with a stored page equal to your screen might look like this:

```
w;;18
```

ATMS-III will respond to the request with **OK**.

## LNGTH Setting

If you are going to edit a document previously stored you may skip this explanation.

The LNGTH (length) default value is normally 100. The current value is displayed at the bottom of the screen after you press the *CLEAR* key located at the top left-hand corner of the keyboard.

If the character string (text unit) you enter contains more characters than the current LNGTH value, ATMS-III will insert a carrier return character at a word break just prior to exceeding the LNGTH value. (Carrier returns must be specifically entered with formatting controls.)

The command to change the LNGTH value is:

```
set;lngth;n
```

where n = any number from 1 to 230.

## Substitute Characters

In the above discussion, carrier return characters were introduced. Display terminals do not provide these characters. ATMS-III provides substitutes for these characters plus the tab and backspace characters. Defaults are provided and they are different for different keyboard types.

The command to determine current substitute characters is:

```
?;set
```

The command to set new substitute characters is:

```
set;subchar;keychar
```

where:

```
subchar = cr (for carrier return)
          tab
          bksp (for backspace)
```

and

```
keychar = any character on your keyboard.
```

For example:

```
set;cr;/
```

makes / the carrier return character.

**Note:** Be careful not to assign substitute characters to characters that are present in the text of your document.

## *Edit Mode Selection*

If you are going to edit a previously stored document, you can skip this explanation.

ATMS-III provides three different edit modes for displaying text to be altered. The three modes are:

| | |
|---|---|
| **script** | To recognize Document Composition Facility (DCF) format controls and Generalized Markup Language (GML) tags and to differentiate between formatted and unformatted types of text for display |
| **atms** | To support documents marked up for the ATMS-III formatter for display purposes |
| **none** | To display documents as entered. |

DCF users should specify **script** edit mode. The current value of edit mode is always displayed in the EMODE field at the bottom of the screen.

The command to set EMODE to SCRIPT is:

```
  set;emode;script
```

## Display Mode Selection

ATMS-III provides several display modes that modify the screen format according to the current EMODE. Two display modes normally are the most important to DCF users—formatted and unformatted.

**Formatted Display Mode:** This means that format mode off text will be displayed as entered with each text unit equal to a display field. Format mode on text will be displayed as paragraphs equal to display fields with line width equal to the WIDTH value at the bottom of the screen.

**Unformatted Display Mode:** This means that all text will be displayed as entered with input units equal to display fields.

The formatted mode is usually more efficient for text alteration.

The current display mode is always displayed in the ALTMODE field at the bottom of the screen.

The command to set display mode for formatted text is:

```
  altmode;f
```

If you wanted the display mode to be unformatted you would use a **u** instead of **f** in the command.

## *Creating a Document*

Once you have checked and/or performed the setup functions, you can begin creation of a new document.

### Initializing Working Storage

Before you enter text of a new document, working storage must be empty. The status of working storage is indicated in the

```
DOC=
```

field at the bottom of the screen. This field contains any of the following:

- Name of document obtained from storage
- Type of document created by ATMS-III—such as report
- No data—indicating text resides in working storage so far unnamed, but created by the operator and not created by ATMS-III
- ws cleared—indicating working storage is currently empty.

So, the command to empty working storage before starting a new document is:

```
clear
```

### Entering Text

After all setup functions and working storage status is as required, you can begin text entry.

Begin by pressing the CLEAR key. The cursor is positioned on line 3. Now you can type your text. Enter substitute carrier returns at the end of DCF control words or control word strings, at the end of text strings followed by a control word, and at the end of text strings to be output as entered.

The cursor will wrap to the next display line.

Transmission to working storage is done with the ENTER key which also blanks the screen for new text entry.

If the screen becomes full and the keyboard inhibited, press RESET followed by ENTER.

If you must see what you just entered, move the cursor to line 1, type the number shown in the PAGE field at the bottom of the screen and ENTER. Then press the CLEAR KEY and resume typing.

### Storing, Replacing, and Deleting Documents

To save the text you have entered in working storage, you must transfer a copy of that text from working storage to permanent storage. To do that you use the "store" command. For instance, you would move the cursor to line 1, then enter:

```
s;dname;ident
```

The response would be:

```
OK, DOCUMENT STORED (dname)
```

**dname** can be any combination of uppercase or lowercase alphabetic or numeric characters and can be up to 15 characters in length. This is the name you are assigning to the document and that you will have to use whenever you wish to retrieve it from permanent storage.

**ident** is an optional item and can be up to 50 characters long. It is an information-only field to further help identify the document. If you do not use *ident*, you do not need the semicolon after *dname*.

Once the document has been stored, you may clear working storage without fear of losing your text. Remember that the store command only stores a copy of working storage; it doesn't clear working storage for you.

If you were to continue working on your document in working storage, you would then replace the copy of the document you previously stored with the one that is the most current.

Use the "replace document" command to replace what you have already stored, such as:

```
rd;dname
```

Response: REPLACE dname;docident ?

Reply:

```
y
```

to complete the replace action.

Notice that you are asked to verify the document involved in the replace action. This is to ensure that you have not made a keying error in the document name. The new document takes the place of the original document, which is deleted (removed) from permanent storage.

Unless you include new identification information (rd;dname;ident), the originally specified information is automatically associated with the newer version of the document in permanent storage. Any response other than y cancels the command.

To delete a document that you no longer need, use the DELETE command, like this:

```
d;dname
```

As for the REPLACE DOCUMENT command, you will be prompted for a "y" reply to complete the action, ensuring that you have properly specified the document name to be deleted.

## Getting a Document

To work on a document previously stored, the document must be in working storage.

The command to get a copy of a stored document into working storage is:

```
g;dname;;t
```

where:

**dname** is as described under "Storing, Replacing, and Deleting Documents" on page 172

and:

**t** restores the initial setup as described under "Setting Up to Enter/Edit a Document" on page 169. (One setup parameter not restored via the **t** is *altmode*. See "Setting Up to Enter/Edit a Document" for altmode explanation.)

Normally, working storage should be empty before copying a stored document into working storage for editing. (See "Creating a Document" on page 172 for working storage status description).

The command to clear working storage and copy a stored document into working storage is:

```
gc;dname;;t
```

If **t** is omitted, the currently set values are used.

## Editing a Document

Once you have the document you want to edit in your working storage, there are a number of ways in which you can make alterations to that document.

### *Finding a Character String*

If you wish to locate an occurrence of a character string, you can use the FIND command.

The command to locate and display a particular character string is:

```
f;character string
```

This form of the command will search from your current working storage location in the document (PAGE field at the bottom of the screen) forward.

The command to locate and display a character string beginning the search at a particular unit of the document is:

```
f;character string;p
```

where p = the first stored unit in the document.

The command to locate the next occurrence after the current location in the stored document is:

```
f;character string;*+1
```

This form of the command will locate and display the next instance of the character string and the command will stay on the command line. Each time you press the ENTER key, the next occurrence of the requested string will be located and displayed.

To locate a character string between two particular units, you can specify:

```
f;character string;p1;p2
```

where:

```
p1 = the unit where you want the search to begin
p2 = the unit where you want the search to end.
```

After displaying the text requested, changes can be made by using Insert Mode, Delete, the EOF key, and by striking over characters.

You will need to move the cursor about on the screen. You can move the cursor up, down, left, or right on the screen by using the four keys showing the arrows that are on the right-hand side of the keyboard.

### *Insert Mode*

If you need to insert text between letters, press the *insert mode* key that appears on the right-hand side of the keyboard. Place the cursor under the letter that you want the inserted text to appear before. Type in the text you want inserted and the text already there will shift to the right. When you have finished inserting text, press the *reset key* that appears in the lower left-hand corner of the keyboard.

### *Delete*

If you need to delete (remove) any characters from a line of text, position the cursor under the first letter of the portion you want deleted and press the *delete key* once for each character you want deleted. The delete key is positioned next to the *insert key.*

### *Erase EOF Key*

There is a key on the far left-hand side of the keyboard that says:

```
ERASE
 EOF
```

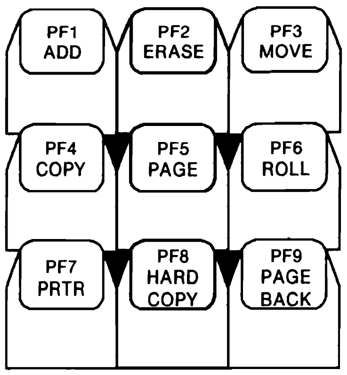
If you position the cursor under a character within a text unit and press the **ERASE EOF** key, the text from the cursor to the end of that text unit will be erased. If you are in formatted display mode (noted in altmode at the bottom of the screen) and the text is format mode on, the **ERASE EOF** key will erase from the cursor to the end of the paragraph.

### *Strikeover*

Anytime that you are entering text onto the screen, whether you're putting in the original text or revising text, you can type over any words you want to change. Simply place the cursor under the first of the group of letters you want to change and type right over them.

## Program Function Key Editing

ATMS-III provides special display terminal functions which are invoked using the PF keys (PFKs). The normal default settings for the PFKs are:



Note that PFKs 1 through 4 actually accomplish editing functions, while the remaining keys are used to move around in your document and cause printouts to occur.

The functions performed by the PFKs will be the same, regardless of which PFK number they are assigned to. If your installation has assigned different numbers than those shown, you should have some type of mask which shows you which function is assigned to which key. If there is none, contact your supervisor or document administrator. The keys will be referenced, from now on, by their functions.

Descriptions of their functions follow.

### *ADD PFK*

This key causes the screen to be split for the addition of text within your current document. To use this function, you first press the ADD PFK, then you place the cursor at the location on the screen after which you want to add text, then press ENTER. ATMS-III will place the first portion of the unit at the top of the screen and the remaining portion of the unit on the bottom, leaving the middle of the screen blank for your text entry.

When you have completed inserting text in this area, no matter how much of the blank area you actually use, press the ENTER key. ATMS-III will include the added text in your working storage document. Empty lines will not be included, so you need not fill the blank area.

### *ERASE PFK*

This key is used to erase data, which can be one or more characters, one or more words, or one or more units. When you press the ERASE PFK, ATMS-III will ask you to position the cursor under where you want the erasure to start. Once you have placed the cursor where the erasure is to start, you press the ENTER key. ATMS-III will then ask you to identify the end of the erasure. Move the cursor under the last position to be erased, then press ENTER. ATMS-III will then do the erasure and re-display the data starting at the noted point.

The erasure need not be limited to what is on the screen, but can be continued to any following point in your document. The PAGE PFK can be used to go forward in your document to locate the point at which you want to end the erasure. If the point is not on a following page, but is on the same page, but not displayed because the page depth is too great for the screen, you use the ROLL PFK to display that point. If the point is not on an adjacent page, a distant page can be accessed if you move the cursor to the PAGE field in the lower right-hand corner, overstrike with the page number desired, and press ENTER.

To use the ROLL PFK, you simply place the cursor at the beginning of a line near the bottom of the screen, then press the ROLL PFK. ATMS-III will then move this line to the top of the screen, displaying the remaining lines of the page which could not fit on one screen image.

If you make a mistake in identifying a location, you can use the PA2 (or FIELD MARK CNCL) key to cancel the command action. This must be done, of course, before you press ENTER for the final location.

### *MOVE PFK*

This key is used to move data within your document. Pressing the MOVE PFK will cause ATMS-III to ask you to identify the location after which you want the data moved. You place the cursor at this point, then press ENTER. ATMS-III will then ask you to identify the position of the first character that you want moved. Again, you place the cursor, then press ENTER. You will then have to identify the location of the last character to be moved. Place the cursor and press ENTER. ATMS-III will accomplish the move operation and display the document starting at the noted point.

Notice you were not told to place the cursor at the beginning or the end of units to do the move. This is because you can move any string of data, even within the same unit, to another location.

To move around in the document while identifying the points involved in the move operation, you use the PAGE or ROLL PFKs to go forward (toward higher page/unit numbers) in your document, or the PAGE BACK PFK to go backward (toward lower page/unit numbers) in your document. If the point is not on an adjacent page, a distant page can be accessed by moving the cursor to the PAGE field in the lower right-hand corner, overstrike with the page number desired, and press ENTER. If you make a mistake, you simply press the PA2 (or FIELD MARK CNCL) key to cancel your command.

### *COPY PFK*

This key works in a manner identical to the MOVE PFK, except that the data between the indicated from and to locations are left in their original locations, as well as being duplicated at the indicated after point.

### *PRINT - This key is used in response to a print command.*

This directs the output of the print command to the printer instead of to your screen. If you do not have a printer assigned, you will be notified. ATMS provides commands for assigning printers.

### *HARD COPY - This key will actually copy the contents*

of a screen, including the status line and any command on the command line, to your local terminal printer. If you do not have a local printer assigned, the command is ignored and you are notified that no printer exists for your terminal.

## Changing Text

You can use the REPLACE TEXT command to change text by entering:

```
r;old;new;nl;n2
```

where:

```
old represents the string of characters
    for which the search is to be made,

new represents the
    string of characters which is to replace each
    occurrence of old

nl represents the unit number at which the
   search is to begin and

n2 represents the unit number at which the
   search is to end.
```

The replace command is a request to find **old** exactly as specified and replace it with **new** exactly as specified. The search continues until:

- **old** has been replaced in 1000 text units
- **n2** has been reached and examined for **old**, or
- the end of the document is reached.

## Re-storing a Document

After editing your document you can choose whether to:

- File a copy in storage with a name different from the original version
- File the current copy in place of the original version using the original version's name.

The command to file in storage with a new name is the same as described under "Storing, Replacing, and Deleting Documents" on page 172.

```
The command to replace an existing copy of the document is
r;dname
```

or, simply:

```
rd; (if the DOC=field at the bottom of the
   screen contains the desired dname).
```

There are two possible responses to this command:

```
DOCUMENT dname DOES NOT EXIST, ENTER 'Y' TO CONTINUE WITH STORE:
```

is issued if the dname you used does not already exist in storage under the opnum indicated.

```
REPLACE dname;ident?
```

is issued when the dname you indicated does currently exist in storage. The ident is the ident currently assigned to the version currently in storage.

You *must reply* to this last response. Press ENTER if the response indicates you've made a mistake in indicating the stored document you want to replace.

Type:

```
y
```

and press ENTER if you want to go ahead with replacing a current stored document with the current contents of working storage.

The response to the entry of "y" is:

```
OK DOCUMENT REPLACED (dname)
```

## *Ending the Session*

When you want to end an editing session you can enter the command:

```
end
```

or, if working storage is not already cleared, enter:

```
end;c
```

which clears working storage and then signs you off.

If the information in working storage is to be kept, you must store it in permanent storage prior to issuing the signoff command.

**Note:** What we have described in this section is one method of creating and editing a file. There is usually more than one way to achieve your desired results. Your location may have procedures in place that differ from what we have described. See your supervisor or document administrator for complete instructions for creating and editing documents at your location.

# Appendix B. A Sample GML Document

On the following pages is a sample document illustrating the use of the Generalized Markup Language starter set. The first part of this appendix shows the text and markup of the document; the second part is the resulting output. We didn't incorporate all of the tags described in this book, but enough of them are shown to give you an idea of how GML works.

**Note:** We used the following SCRIPT command options to run our sample document:

```
MESSAGE(I D) TW SYS(S O) INDEX
```

```
:gdoc sec='Company Confidential'.
:frontm.
:titlep.
:title stitle='A Short GML Document'.
A Short GML Document Demonstrating the Document Composition Facility
:date.
:author. Barb Dwyer
:address.
:aline.77 Cattle Drive
:aline.Silverado, Colorado
:eaddress.
:etitlep.
:toc.
:figlist.
:tlist.
:body.
:h1.Introduction to the Generalized Markup Language
:ih1 id=gml.Generalized Markup Language
:i2.Introduction to
:ih1 seeid=gml.GML
:p.The practice of generalized markup of source documents has
significant benefits not only for SCRIPT/VS processing,
but also facilitates many other applications.
:h2.What Is Generalized Markup?
:p.To mark up a source document is to add information to it that
:ih1 id=mark.markup
:i2.definition of
enables a person or system to process it in some way.
The added information or markup can be instructions called
control words, or indications, or descriptions of some kind.
Document markup is the primary means of instructing a
computerized text processing system, such as SCRIPT/VS,
how a document is to be processed.
:p.A document can be marked up in two ways:
:ol.
:li.With specific markup, which limits text processing
:i2.specific
to a particular application.
:li.With generalized markup, which describes the contents
:i2.generalized
of a source document, without regard to particular processing.
The language used for generalized markup (GML, for Generalized
Markup Language) is descriptive; it provides the syntax and
usage rules for developing your own vocabulary of tags
for :hp2.describing:ehp2.
:i1.tags :i2 refid=gml.tags
the parts of a document.
:fn.
A starter set of GML tags is provided with
the Document Composition Facility to allow the user to get going.
:efn.
:eol.
:p.A GML tag identifies the associated text as a
particular document element.
For example:
```

```
:ul.
:li.A book might have the major divisions:
:ul.
:li.front matter
:li.body
:li.back matter
:eul.
:p.Within those divisions we can have paragraphs,
examples, figures, list items, and so on.
:li.A memo might have document elements of addressee,
date, sender, address, subject, and reference, as well as
other types similar to those of a book.
:eul.
:p.You might ask, :q.Why use generalized markup instead
of specific markup:eq.?
Well, thinking in terms of the content or purpose of
the parts of a document is easier than thinking in terms of how
the parts should be processed or how they should appear when
printed.
A person doing generalized markup can concentrate on the
text, without thinking about format.
:p.There are benefits of generalized markup when your only
application will be text processing with SCRIPT/VS and others
when you anticipate additional applications.
:fig id='abc' width='4i' depth='10p6' frame='box'.
I've just drawn a little figure here
where I've left 10 picas and 6 points of
space before the text of the figure
started.  I've set my width to four
inches -- if I print out my
document with a column width of
four inches or larger, my figure
will be column width, if I print
out my document with a column width
of less than four inches, my figure
will extend the width of the page
dimension.
:figcap.A Sample Figure
:figdesc.This figure also gives you a little bit of information
on how the WIDTH attribute of the FIG tag works.
:i1.figure example
:i2 refid=mark.of a figure
:efig.
:h3.Potential Benefits of GML for SCRIPT/VS Text Processing
:p.Here are some potential benefits in using GML when you intend
to use SCRIPT/VS only for text processing:
:ul.
:li.Alternative GML interpretation.
A GML tag need not be limited to a single SCRIPT/VS
interpretation.
For example, a tag might indicate that a group of
words in the text is a title.
For one application, you might want titles to be enclosed
in quotation marks.
But for another application, you might want titles
```

```
to be underlined or set in italics.
Each application could be satisfied by alternative GML
interpretations, :hp1.with no change:ehp1. to the source
document or to the markup of titles.
:i2 refid=gml.potential benefits of
As described in a later section in this chapter, you
have control over the way GML is interpreted.
:li.Ease of markup.
It is easy to remember GML tags because they can
consist of terms and abbreviations commonly
used to describe a document.
GML generally requires fewer characters to be entered
for markup than a corresponding complex sequence of
control words.
The result is faster markup and keying of the document.
Changes to the markup are also faster and may even be
eliminated with GML, because you can control how GML
is interpreted.
:li.Ease of text update.
It's easy to update text marked up with GML.
With GML, such things as the numbering of items in a
numbered list is left to the formatter, which numbers
the items automatically.
Thus, when you insert or delete an item, you don't
have to renumber subsequent items, as you would have to
if the numbers were part of the source text.
:eul.
:rdef id=corptbl cwidths='1i * * * * *'
arrange='1 2 3 3 3 3'
arrange='1 2 4 5 6 6'
arrange='1 2 4 5 7 8'
align='left left center center center left center'
concat='yes yes no yes'
valign='center center top center center top bottom'.
:table refid=corptbl id=tbtest.
:i1.table example
:i2 refid=mark.of a table
:row.
:c.Selected major industry and selected country
:c.Number of U.S. corporation returns
:c.Controlled Foreign Corporations
:c.Number of foreign corporations
:c.Total assets of corp.
:c.Foreign corporations with current earnings and
profits (+) before taxes
:c.Current
earnings and
profits before taxes
:c.Foreign income taxes (net)
:erow.
:tcap.A Sample Table
:etable.
:backm.
:index.
:egdoc.
```

# A SHORT GML DOCUMENT FOR DEMONSTRATING THE DOCUMENT COMPOSITION FACILITY

*February 6th, 1987*

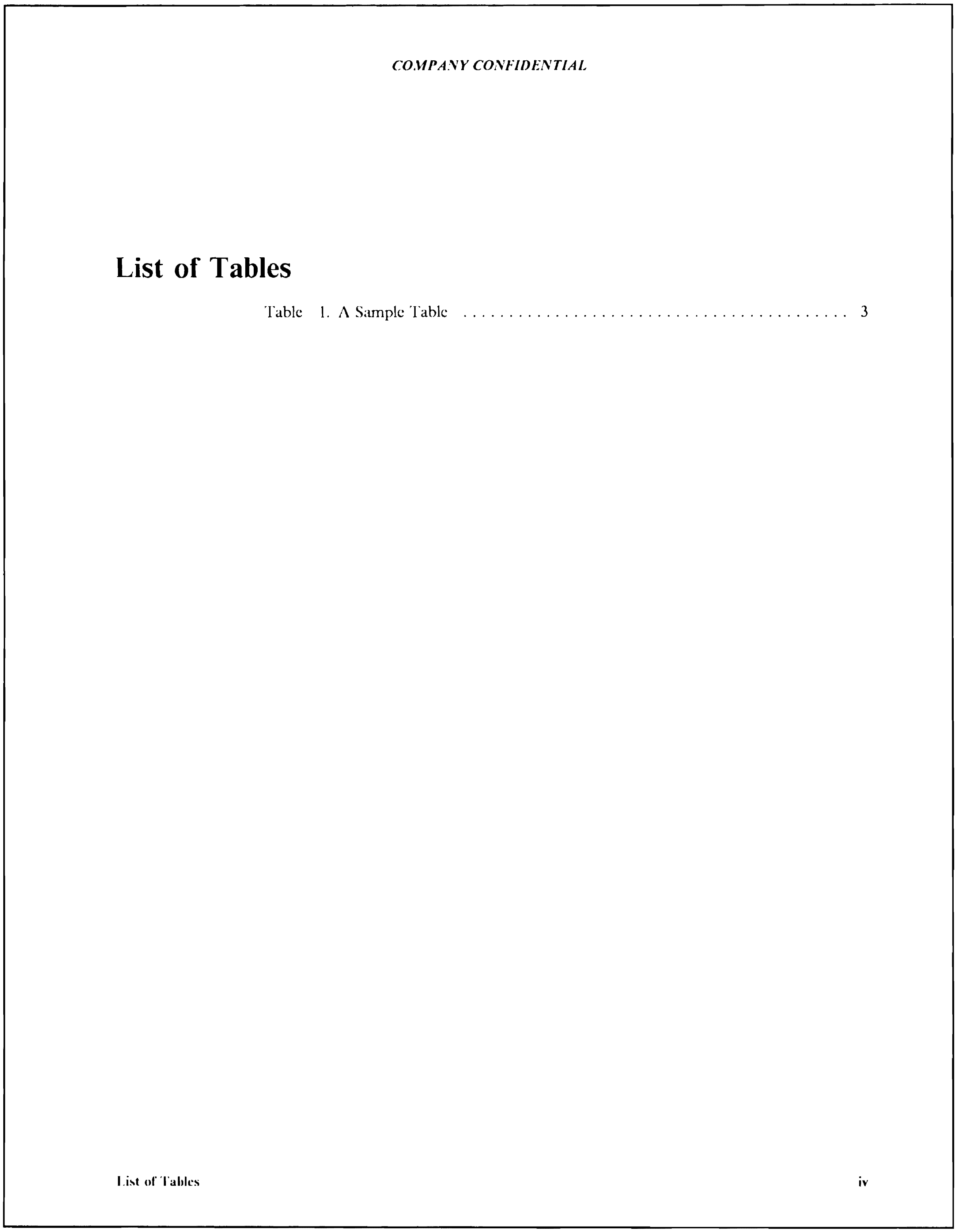
Barb Dwyer

77 Cattle Drive
Silverado, Colorado

***Company Confidential***

# Table of Contents

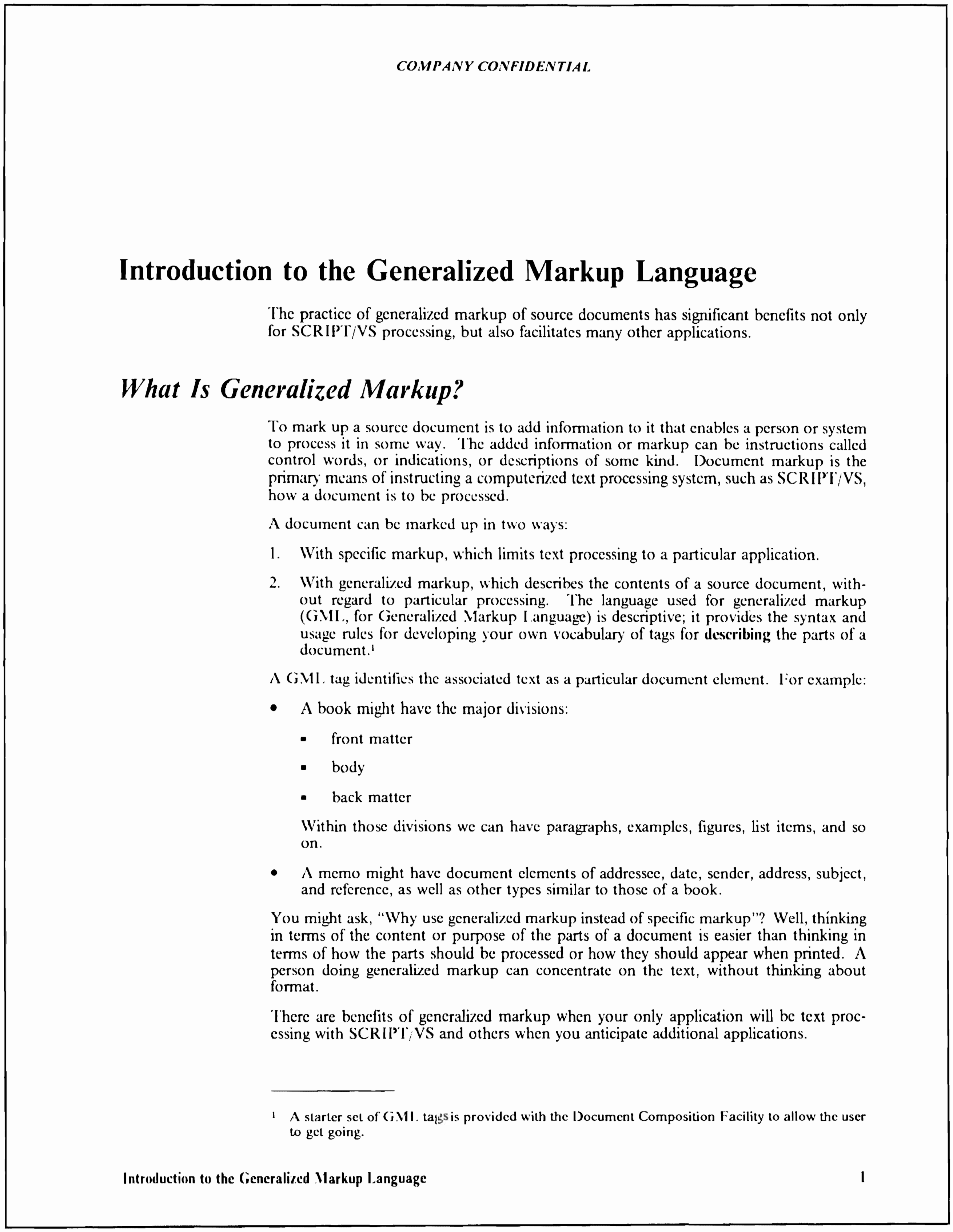
## List of Illustrations



COMPANY CONFIDENTIAL

## List of Tables

List of Tables iv



*COMPANY CONFIDENTIAL*

# Introduction to the Generalized Markup Language

The practice of generalized markup of source documents has significant benefits not only for SCRIPT/VS processing, but also facilitates many other applications.

## *What Is Generalized Markup?*

To mark up a source document is to add information to it that enables a person or system to process it in some way. The added information or markup can be instructions called control words, or indications, or descriptions of some kind. Document markup is the primary means of instructing a computerized text processing system, such as SCRIPT/VS, how a document is to be processed.

A document can be marked up in two ways:

1. With specific markup, which limits text processing to a particular application.
2. With generalized markup, which describes the contents of a source document, without regard to particular processing. The language used for generalized markup (GML, for Generalized Markup Language) is descriptive; it provides the syntax and usage rules for developing your own vocabulary of tags for **describing** the parts of a document.[1]

A GML tag identifies the associated text as a particular document element. For example:

- A book might have the major divisions:
  - front matter
  - body
  - back matter

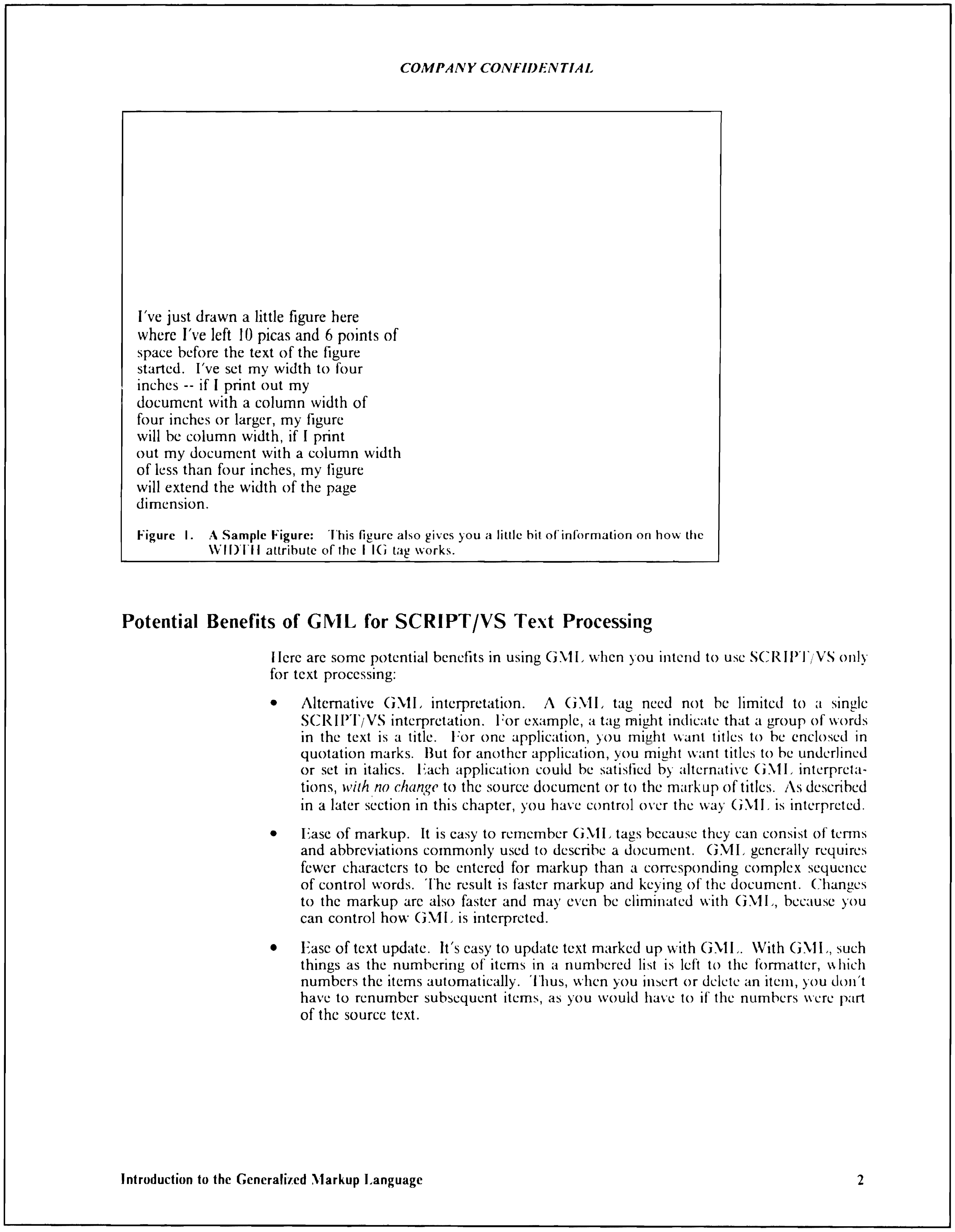
  Within those divisions we can have paragraphs, examples, figures, list items, and so on.
- A memo might have document elements of addressee, date, sender, address, subject, and reference, as well as other types similar to those of a book.

You might ask, "Why use generalized markup instead of specific markup"? Well, thinking in terms of the content or purpose of the parts of a document is easier than thinking in terms of how the parts should be processed or how they should appear when printed. A person doing generalized markup can concentrate on the text, without thinking about format.

There are benefits of generalized markup when your only application will be text processing with SCRIPT/VS and others when you anticipate additional applications.

[1] A starter set of GML tags is provided with the Document Composition Facility to allow the user to get going.

**Introduction to the Generalized Markup Language** **1**



*COMPANY CONFIDENTIAL*

I've just drawn a little figure here
where I've left 10 picas and 6 points of
space before the text of the figure
started. I've set my width to four
inches -- if I print out my
document with a column width of
four inches or larger, my figure
will be column width, if I print
out my document with a column width
of less than four inches, my figure
will extend the width of the page
dimension.

**Figure 1. A Sample Figure:** This figure also gives you a little bit of information on how the WIDTH attribute of the FIG tag works.

## Potential Benefits of GML for SCRIPT/VS Text Processing

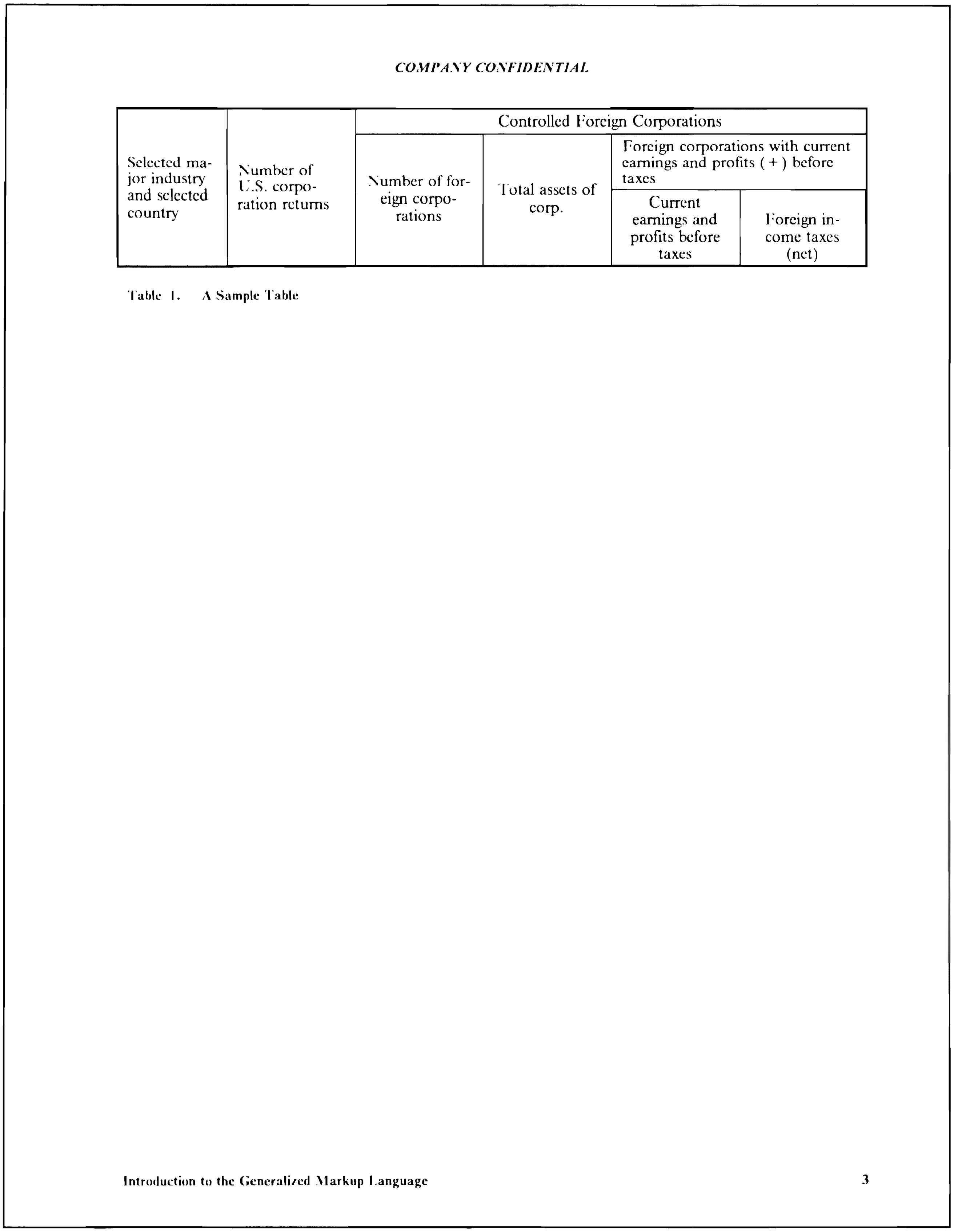
Here are some potential benefits in using GML when you intend to use SCRIPT/VS only for text processing:

- Alternative GML interpretation. A GML tag need not be limited to a single SCRIPT/VS interpretation. For example, a tag might indicate that a group of words in the text is a title. For one application, you might want titles to be enclosed in quotation marks. But for another application, you might want titles to be underlined or set in italics. Each application could be satisfied by alternative GML interpretations, *with no change* to the source document or to the markup of titles. As described in a later section in this chapter, you have control over the way GML is interpreted.

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

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

**Introduction to the Generalized Markup Language** **2**



*COMPANY CONFIDENTIAL*

| Selected major industry and selected country | Number of U.S. corporation returns | Controlled Foreign Corporations | | | |
|---|---|---|---|---|---|
| | | Number of foreign corporations | Total assets of corp. | Foreign corporations with current earnings and profits (+) before taxes | |
| | | | | Current earnings and profits before taxes | Foreign income taxes (net) |

Table 1. A Sample Table

Introduction to the Generalized Markup Language 3

# Index

# Appendix C. Solutions for Exercises

## *Markup for Getting Started—Paragraphs and Headings*

The following markup is for "Exercise: Paragraphs and Headings" on page 13.

```
:h3.This is a Head Three
:p.It has two short paragraphs
following it.
This is the first paragraph.
Each sentence was started on a
new line.
:p.This is the second paragraph.
Look to see how DSMPROF3 has
formatted it (a skip before,
and no indention).
:h5.This is a Head Five
:p.The paragraph following it
begins on the same line as the
head, even though it was entered
on a separate line.
:p.The next paragraph under the
head five formats like this.
```

## Markup for Lists

The following markup is for "Exercise: Lists" on page 26.

```
:p.Please send the following items:
:dl tsize='11'.
:dthd.Number
:ddhd.Description
:dt.A107C2
:dd.Class A Widget, 100 x 200 mm
in assorted colors (no white).
:dt.B321DJ
:dd.Brown and yellow Whuzzit,
with linkage for optional nutcracker
attachment.
:dt.BY7532
:dd.Assorted gimcracks, with the
following characteristics:
:ul.
:li.Soft
:li.Cuddly.
:eul.
:edl.
:p.When I get around to it, I will
:ol.
:li.Balance my checkbook.
:li.Prune the shrubbery.
:li.Write to the following:
:sl compact.
:li.My mother in New Rochelle
:li.My great aunt in Detroit
:li.My niece in Ossining
:li.My friend in South Bend.
:esl.
:li.Clean out the garage.
:eol.
```

## *Markup for Examples and Figures*

The following markup is for "Exercise: Examples and Figures" on page 38.

```
:p.Here's an example of some BASIC statements
:xmp.
10 PRINT USING 55 A, B, C
20 LET J = K + 2
30 IF J = X GO 80
:exmp.
:pc.that will solve this problem.
:p.An inline, page-wide figure would look like this:
:fig place=inline width=page frame=box.
A PAGE-WIDE FIGURE
  Since the contents of a figure
  are formatted EXACTLY as entered,
  you can enter blanks
  on the line        (before text)
      and the lines
      will print
          Exactly
          the same
               as the way
               they were entered!
:figcap.A Page-Wide Figure
:figdesc.This is the first figure I have
entered myself.
:efig.
:p.This paragraph follows the FIG end tag.
Here we have another figure (inline and column-wide):
:fig place=inline width=column.
Now let's create another figure. This one is column-wide.
(This exercise will also create a second item for our listing,
in a future exercise involving a List of Illustrations.)
:figcap.This Is A Column-Wide Figure.
:efig.
```

## Markup for Tables

The following markup is for " Exercises: Tables" on page 60.

### First Table Exercise

```
:rdef id=autohdr cwidths=',5i * * *' valign=bottom
      hp='2 2 2 3' align='left center center right'.
:rdef id=auto cwidths=',5i * * *' align='left right'
:table column refid=auto width='3i'
:thd refid=autohdr.
:c.Year:c.Passenger Cars:c.Trucks and Buses:c.Total
:ethd.
:row refid=auto.
:c.1900:c.4,192:c 4.4,192
:c.1905:c.24,250:c.750:c.25,000
:c.1910:c.181,000:c.6,000:c.187,000
:c.1915:c.895,930:c.74000:c.969,930
:c.1920:c.1,905,560:c.321,789:c.2,227,349
:tcap.U.S. Automobile Production
:tdesc.The above table shows the number of automobiles
produced in the United States from the years 1900 through 1920.
:etable.
```

### Second Table Exercise

```
:rdef id=flag arrange='1 2 / 1 3 / 1 4 / 1 5 / 1 6 / 1 7 / 1 8 '
              arrange='9 9 / 10 10 / 11 11 / 12 12 / 13 13 / 14 14'
              align='c r' valign='c t'.
:table column refid=flag.
:row.
:c.Flag
:c.Delaware:c.Connecticut:c.Maine:c.Maryland:c.Massachusetts
:c.New Hampshire:c.New Jersey:c.New York:c.North Carolina
:c.Pennsylvania:c.Rhode Island:c.Vermont:c.Virginia
:etable.
```

## *Markup for Cross-References and Footnotes*

The following markup is for "Exercise: Cross-References and Footnotes" on page 69.

```
:h4 id=exh4.Heading Being Referred To
:p.This example will show cross-references to:
:ol.
:li.A figure
:li.A footnote
:li id=three.A list item
:li.A table
:li.A heading.
:eol.
:p.Here's a cross-reference to a figure that
comes later, so we need to run with the
FPASSES or the TWOPASS option to see the correct results:
see :figref refid=exf4..
:fn id=exer4.
Of course, when you run the exercise, the footnote will
be numbered 1, since it is the first one in the
document.
:efn.
:p.Here we have a footnote
reference.:fnref refid=exer4.
:fig place=inline id=exf4 frame=box width=18p.
This figure needs referring to.
:figcap.Figure Being Referred To
:efig.
:p.Don't forget to put an ID attribute on number
:liref refid=three. so that this reference to it resolves properly.
:p.We also want to refer to a table. So, see
:tref refid=jb..
:rdef id=xwins align=center cwidths='2i 1i 1i'.
:table column refid=xwins id=jb width='4i'.
:row.
:c.Here's
:c.the
:c.table
:row.
:c.being
:c.referred
:c.to
:tcap.This part of the exercise is a table
:etable.
:p.And finally, we have a heading reference;
see :hdref refid=exh4..
:p.(Before we finish this exercise,
we must also check to see that
all our punctuation is correct.)
```

## *Markup for Highlighting, Citing, Noting, and Quoting*

The following markup is for "Exercise: Highlighting, Citing, Noting, and Quoting" on page 75.

```
:p.Speak :hp2.boldly:ehp2. on
the 3800, :hp2.strikingly:ehp2. on the 1403.
:p.Title references are :hp1.also:ehp1.
used for movie titles, such as
:cit.Casablanca:ecit..
Humphrey Bogart is often misquoted
as having said :q.Play it again,
Sam:eq. in that movie.
I wish people would learn to
quote :hp3.accurately:ehp3..
:p.If I said, :q.he said :q.yes:eq.
to me,:eq. that would be an example
of a nested quote.
:p.Look at this excerpt
from the :cit.GML User's Guide:ecit.:
:lq.
:p.You may well ask, :q.Why
use quote tags at all?:eq.
:p.We now have a variety of output devices
on which to print our documents.
Some of these devices print using
fonts that distinguish among open quotation marks,
close quotation marks, and apostrophes.
By using the proper markup, our files
can be printed on any of the
available devices with no change to the markup.
:elq.
:note.Notes lose their impact if used excessively.
```

# Appendix D. Differences from Earlier Releases

For those of you who are familiar with the starter set as it was made available in the earlier releases of the Document Composition Facility, this appendix summarizes the new features available with this release of the Document Composition Facility.

Central Programming Service support and maintenance are available for the starter set. Support and maintenance *are not* available, however, if the starter set has been modified in any way. If you do make modifications, it is recommended that you also maintain an *unmodified* starter set for diagnostic purposes.

Starter set tags are fully documented in the *Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide.* This book also discusses ways in which the GML starter set may be modified.

## *Release 3.1 Changes*

Release 3.1 makes the following GML improvements:

- You can build tables with new table tags. Various attributes are provided to give you control over aspects of table formatting such as cell width, text alignment, and rotation. For more information, see "Chapter 5. Creating Tables" on page 41.
- You can now request *more than two* formatting passes. For more information, see "FPASSES(n)" on page 121.
- The sort sequence for special characters and accented characters in index entries has been changed.

## *Release 3 Changes*

Changes made to the Release 3 starter set included:

- All macros were renamed to begin with the characters "DSM."[33]
- Comments in the profile and macros were reworked to more fully explain tag processing.
- Leading blanks are never discarded; therefore, all lines in the source document should begin at the left margin.
- The "special characters" header in the index was removed.
- Support for page printers.
- Heading definitions were no longer dependent on the page layout. (In Release 2, spacing around headings changed if you used offset style [SYSVAR "S"]).

[33] This change allowed Central Programming Service support for the starter set.

- Heading levels 1 and 0 (:H1 and :H0) were no longer out-justified when you duplex in offset style.
- Title page spacing was different for Release 3.
- Several changes were made to correct starter set footnoting problems. One of these changes resulted in footnotes no longer being indented on the right.
- A footnote reference was automatically inserted into the document when a footnote was defined without specifying a unique identifier with the ID attribute. (This reduced the amount of markup required for most footnotes.)
- The imbed trace was printed in a single column. (In Release 2, the imbed trace printed in two-column format.)
- In Release 3, the line length value for two-column format was 6.8 inches. (The line length value for two-column format was 6.67 inches in Release 2.)
- All undocumented tags and attributes were removed from the starter set.
- The text specified on the SEC attribute of the GDOC tag appeared in highlight level 2 on the title page and as a running heading on the top of each page.
- The punctuation following the end tag for quotations (EQ) was made to appear in the printed document wherever the punctuation is placed in the input document. Punctuation was no longer moved inside of quotation marks.
- The tags for marking-up glossary lists, which had been in the IVPROF2 profile, were included as part of the starter set.
- The continuation character was changed from "+" to hexadecimal "03."
- The default profile for Release 3 is named "DSMPROF3."
- The default macro library for Release 3 is named "DSMGML3."

Additionally, the .SI [Segment Include] control word was described. (This control word allows images to be included with the text on the page when printing on page printers.)

## New and Extended Tags and Attributes

- New tags and attributes have been added to work within the definition list (DL).

  The tags are

  - the DTHD (definition term heading) tag which allows you to specify a heading for the terms in a definition list
  - the DDHD (definition description heading) tag which allows you to specify a heading for the descriptions in a definition list.

  The attributes are

  - BREAK, a value attribute that causes a break after a long definition term so that the definition description begins on the following line, rather than the same line
  - HEADHI, a labeled attribute that indicates the level of highlighting you want for the definition term and definition description headings.

- An ID attribute can be added to any list item to name that list item so that you can refer to it from some other place within a document.

  A new tag, LIREF (list item reference), allows you to refer to any list item that you have identified by name with the ID attribute.

## Run-Time Options (SCRIPT Command Options)

This list includes only those items that are relevant to typical GML processing. For complete information on new run-time capabilities, see the *Document Composition Facility: SCRIPT/VS Language Reference.*

- Three page devices have been added as allowable printers. They are the IBM 4250 printer, the IBM 3800 Printing Subsystem Model 3, and the IBM 3820 Page Printer. You can specify one of the new logical device types on the DEVICE option of the SCRIPT command.

| PAGE DEVICE | PAGE WIDTH | LENGTH |
|---|---|---|
| 4250A | 8.5i | 11i |
| 4250B | 11 i | 17i |
| 4250L | 8.5i | 14i |
| 4250A3 | 297mm | 420mm |
| 4250A4 | 210mm | 297mm |
| 38PPN | 8.5i | 10i |
| 38PPW | 13.5i | 10i |
| 38PPNS | 11 i | 7.5i |
| 38PPWS | 13.5i | 7.5i |
| 38PPW90 | 10 i | 13i |
| 38PPNS90 | 7.5i | 11i |
| 38PPW270 | 10 i | 13.5i |
| 3820A | 8.5i | 11i |
| 3820A90 | 11i | 8.5i |
| 3820A180 | 8.5i | 11i |
| 3820A270 | 11i | 8.5i |
| 3820L | 8.5i | 14i |
| 3820A4 | 210mm | 297mm |
| 3820B4 | 257mm | 364mm |
| 3820B5 | 182mm | 257mm |

- You can change the typeface used for the whole document by specifying a coded font on the CHARS option of the SCRIPT command when formatting for page printers. See your supervisor or document administrator for the fonts that are available in your organization.
- The FONTLIB option specifies the font library to be used for page printers.
- There are two additional SYSVAR options, R (for READ) and W (for WRITE). These SYSVAR options are valid *only* in CMS and TSO.

# Glossary

The glossary defines words and phrases that have special meanings in SCRIPT/VS or special meanings in a typographical sense. The terms are defined as they are used in this book. If you do not find the term you are looking for, refer to the index or to the *IBM Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

This glossary includes definitions developed by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). This material is reproduced from the American National Dictionary for Information Processing, copyright 1977 by the Computer and Business Equipment Manufacturers Association, copies of which can be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

**advanced function printing (AFP):** The ability of licensed programs to use the all-points-addressable concept to print text and images on a printer.

**alignment:** The horizontal placement of text in a column or cell.

**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.

**alphameric string:** A sequence of characters consisting solely of the letters a through z and the numerals 0 through 9.

**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). In running footings, running headings, and running titles, the ampersand is usually the page number symbol. When encountered by itself on the right side of a .SE [Set Symbol] control word, it is interpreted as the page number symbol.

**application processing function (APF):** In GML processing, the processing that is performed when a document element or attribute is recognized. In SCRIPT/VS, an APF is implemented as a sequence of control words, possibly intermixed with text and symbols, in one of three forms: macro definition, value of a symbol, or imbedded file.

**application programming interface (API):** An external, published interface that can be programmed to by another application.

**ascender:** (1) In a font, the distance from the baseline to the top of the character. See *maximum ascender.* (2) The part of a lowercase letter that rises above the body of the letter. Letters with ascenders are b, d, f, h, k, l, and t.

**ATMS-III:** Advanced Text Management System.

**attribute:** A characteristic of a document (or document element) other than its type or content. For example, the security level of a document or the depth of a figure.

**attribute label:** In GML markup, a name of an attribute that is entered in the source document when specifying the attribute's value.

**back matter:** In a book, those sections (such as glossary and index) that are placed after the main chapters or sections.

**balancing:** In multicolumn formatting, the process of making column depths on a page approximately equal by re-distributing the text in the columns. See also *vertical justification.*

**baseline:** An imaginary horizontal line that most of the letters in a line of text appear to rest on.

**basic document element:** In a general document, one of a group of elements that occurs frequently; for example: note, paragraph, and definition list.

**batch environment:** The environment in which non-interactive programs are executed.

**binding edge:** The edge of a page to be bound, stapled, or drilled. Defined with the BIND option of the SCRIPT command.

**body:** (1) Of a printed page, that portion between the top and bottom margins that contains the text. (2) Of a book, that portion between the front matter and the back matter.

**boldface:** A heavy-faced type. Also, printing in this type.

**bottom margin:** On a page, the space between the body or the running footing, if any, and the bottom edge of the page.

**break:** An interruption in the formatting of input lines so that the next input line is printed on a new output line.

**bug:** An error in a program on in a document markup.

**call:** Used in reference to macros. It means to invoke the macro.

**caps:** Capital letters. See also *initial caps.*

**caption:** Text accompanying and describing 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.

**CDPF:** Composed Document Printing Facility.

**cell:** A single unit within a table in which text or other expressions may appear. A cell is always rectangular and usually bounded by horizontal and vertical rules.

**centimeter (cm):** A measurement equal to 0.39 inch. 100 cm = 1 meter (m).

**chapter:** In a general document, a name given to a first-level heading segment that occurs within the body of a document. See also *heading segment.*

**character:** A symbol used in printing. For example, a letter of the alphabet, a numeral, a punctuation mark, or any other symbol that represents information.

**character arrangement table:** Translates input data into printable characters and identifies associated character sets and graphic character modification modules.

**character set:** A finite set of different characters that is agreed to be complete for some purpose. For example, in printing, the characters that constitute a font.

**character space:** The horizontal size of a character. This size depends upon which font the character is from and on which physical device the character is printed.

**character spacing:** The space between characters in a word.

**cicero:** In the Didot point system, a unit of 0.1776 inch (4.512 millimeters) used in measuring typographical material.

**CMS:** Conversational Monitor System&em.an interactive processor that operates within VM/370.

**code page:** A font library member name that gives the association between code points and the character names of a font.

**code point:** An eight-bit binary code representing one of 256 possible characters.

**coded font:** (1) The combination of a code page and a font library. (2) A font that is fully described in terms of typeface, point-size, weight, width, and attribute.

**column:** A vertical arrangement of characters or other expressions on a printed page.

**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 width:** The width of each text column on a page. Specified with the .CL [Column Line Length] control word. (In multicolumn formatting, all columns on a page usually have the same width.)

**command:** A request from a terminal or a request specified in a batch processing job for the performance of an operation or the execution of a particular program. For example, a request given at a terminal for SCRIPT/VS to format a document or for an editor to edit a line of text.

**comment:** A control word line that is ignored by SCRIPT/VS. Such lines begin with either .* or .cm.

**composed text:** Text that has been formatted and that contains control information to direct the presentation of the text on page printers.

**composite:** The act or result of formatting a document.

**composite rotation:** The total amount of rotation done by the printer to place text in the correct orientation on the page.

To determine the composite rotation, add all the current rotations. Such as the rotation of the page as specified with the DEVICE command option, the rotation of the current area (as specified on the .DA [Define Area] control word), the rotation of the current table (as specified in the TABLE parameter of the .TD [Table Definition] control word), and the rotation of the contents of the current cell (as specified in the CELL parameter of the .TD [Table Definition] control word). For example: if the page rotation is 90 and there is no current area, and the table rotation is 180, and the rotation of the contents of the cell is

180; then the composite rotation of the table would be 270 and the composite rotation of the contents of the cell would be 90.

**compositor:** A person or program that composes text.

**concatenation:** The forming of an output line that contains as many words as the column width allows, by placing the first words from an input line after the last words from the preceding input line. When words from an input line would reach beyond the right margin and hyphenation cannot be performed, they are placed at the beginning of the next output line, and so on.

**control word:** An instruction within a document that identifies its parts or tells SCRIPT/VS how to format the document. See also *macro.*

**control word line:** An input line that contains at least one control word.

**control word statement:** A control word and its parameters.

**copy group:** A portion of a FORMDEF that defines a set of modifications that can be used when printing a page.

**current left margin:** The left limit of a column that is in effect for formatting. Each column's left margin is specified with the .CD [Column Definition] control word. However, the current left margin (that is, the left boundary for an output line) might vary to the right of the column's left margin when indention is changed with the .IN [Indent], .UN [Undent], .IL [Indent Line], and .OF [Offset] control words.

**current line:** The line in a source document at which a computer program (such as an editor or a formatter) is positioned for processing.

**debug:** To detect, trace, and eliminate errors in computer programs and SCRIPT/VS documents.

**default value:** A value assumed by a computer program when a control word, command, or control statement with no parameters is processed. In GML processing, the value assumed for an attribute when none is specified.

**descender:** (1) In a font, the distance from the baseline to the bottom of the character. See *maximum descender.* (2) The part of a letter that falls below the body of the letter. Letters with descenders are g, j, p, q, y, and Q.

**destination:** The physical device to which data is sent.

**dictionary:** A collection of *word stems* that is used with the spelling verification and automatic hyphenation functions.

**Didot point system:** A standard printer's measurement system on which type sizes are based. A Didot point is 0.0148 inch (0.376 millimeter). There are 12 Didot points to a cicero. See also *cicero* and *point.*

**document:** (1) A publication or other written material. (2) A machine-readable collection of lines of text or images, usually called a source document. See also *output document* and *source document.*

**document administrator:** One who is responsible for defining markup conventions and procedures for an organization.

**document conversion processor:** A computer program that processes a machine-readable document that includes formatting controls written in one formatter language, to produce a machine-readable document that includes formatting controls appropriate for another formatter language.

**document library:** A set of VSAM data sets, accessible in a batch environment, which contain documents and related files.

**dot leader:** A set of periods that fills in the space between two pieces of split text such as a chapter title and its page number in a table of contents.

**duplex:** A mode of formatting appropriate for printing on both sides of a sheet.

**EBCDIC:** Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters.

**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 usually equal to the width or the height of the character "m" in a particular font.

**en:** A unit of measure usually equal to one-half the width of an em. For many typefaces, lowercase characters tend to average the width of an en.

**enable:** Used in reference to a tag. Means that the tag is mapped to its appropriate APF.

**epifile:** The second portion of a profile (after a .EF control word) that is processed *after* a main document has been processed.

**escapement:** The space unit of movement (either vertical or horizontal) that is built into a physical device. For the 1403. with a 10-pitch train, the horizontal escapement unit is 1/10th of an inch; for the 4250, that value is 1/600th of an inch; and for the 3800 Model 3 and the 3820 Page Printer, that value is 1/240th of an inch.

**extended symbol processing:** The processing of a symbol whose value causes the remainder of the line to be stacked and later processed as a new input line.

**factor:** A dimensionless scalar value used to form a product with another value. Factors can also be expressed as percentages.

**figure space:** (1) The width of the figure zero (0) is commonly used as the figure space of a given typeface. This is the definition figure space as it is used in DCF. (2) A unit of measure equal to the width of the "en" space in a particular font.

**fill character:** The character that is used to fill up a space; for example, blanks used to fill up the space left by tabbing.

**float:** (1) (noun) A keep (group of input lines kept together) whose location in the source file can vary from its location in the printed document. (2) (verb) Of a keep, 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.

**folio:** Page number.

**font:** (1) An assortment of type, all of one size and style. (2) A font library member that contains characters that must be used in conjunction with a code page font library member.

**font object:** Refers to a member of a font library. In CMS, a font object is a file whose filetype matches the name of the font library. In MVS, a font object can be a member of a partitioned data set (PDS).

**font set:** The set of fonts to be used in formatting a source document.

**footing:** Words located at the bottom of the text area. See also *running footing*.

**footnote:** A note of reference, explanation, or comment, placed below the text of a column or page, but within the body of the page (above the running footing).

**foreground:** The environment in which interactive programs are executed. Interactive processors reside in the foreground.

**format:** (1) (noun) The shape, size, and general makeup of a printed document. (2) (verb) To prepare a document for printing in a specified format.

**formatter:** (1) A computer program that prepares a source document to be printed. (2) That part of SCRIPT/VS that formats input lines for a particular logical device type.

**formatting mode:** In document formatting, the state in which input lines are concatenated and the resulting output lines are justified.

**form definition:** A resource object that defines the characteristics of the form which include: overlays to be used (if any), text suppression, the position of page data on the form, and the number and modifications of the page. Synonymous with FORMDEF.

**FORMDEF:** Synonym for form definition.

**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 that can be used to identify the parts of a source document without respect to particular processing.

**GML delimiter:** A special character that denotes the start of GML markup. In the starter set, it is initially a colon (:).

**GML end tag delimiter:** A special character that denotes the end of GML markup. In the starter set, it is initially a period (.).

**GML interpretation:** Interpretation of GML markup consists of recognizing the start or end of an element (or an attribute label), associating it with an APF, and executing the APF. In SCRIPT/VS, interpretation is performed jointly by SCRIPT/VS itself and by APFs.

**graphic character modification (GCM) module:** Modules that contain scan patterns of IBM-supplied character sets and/or user-defined character sets, without respect to particular processing.

**Graphical Data Display Manager (GDDM):** An IBM licensed program that creates page segments.

**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 spaces). 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.*

**heading segment:** An element that begins with a heading, followed by basic document elements and lower-level heading segments.

**hexadecimal:** Pertaining to a number system based on 16, using the sixteen digits 0, 1, . . . 9, A, B, C, D, E, and F. For example, hexadecimal 1B equals decimal 27. See also *EBCDIC.*

**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 a 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.

**IEBIMAGE:** Utility program that creates and maintains various 3800 Printing Subsystem Model 1 modules (for example, character arrangement table and graphic character modification (GCM) modules) and stores them in SYS1.IMAGELIB.

**image:** A likeness or imitation of an object, such as a picture or logo.

**impact printer:** A printer, such as the 1403 and the 3211, in which printing is the result of mechanical impacts.

**implied paragraph structure:** An element that begins with an implied paragraph; that is, one for which you do not specifically enter a paragraph tag. The existence of the paragraph is understood from the existence of the implied paragraph structure, for example, as with notes, figure captions, and footnotes.

**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], .UN [Undent], .OF [Offset], and .IL [Indent Line] 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.

**initial value:** A value assumed by SCRIPT/VS for a formatting function until the value is explicitly changed with a control word. The *initial value* is assumed even before the control word is encountered, whereas the *default* value is assumed when the control word is issued without parameters. See also *default value.*

**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.

**inline space:** An amount of horizontal white space in a line that usually occurs between two words.

**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 formatter.

**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.

**intercharacter space:** Extra horizontal white space inserted *between* characters of a word. This space is in addition to the space included as part of the characters by the designer of the font.

**interword space:** See *word space.*

**Interactive System Productivity Facility (ISPF):** A dialog manager for interactive applications that provides control and services to allow processing of the dialogs in different host environments.

**italic:** A typestyle with characters that slant upward to the right.

**job control language (JCL):** A language of job control statements used to identify a computer job or describe its requirements to the operating system.

**job control statement (JCS):** A statement that provides an operating system with information about the job being run.

**justification:** See *horizontal justification* and *vertical justification.*

**justify:** (1) (ISO) To control the printing positions of characters on a page so that the left-hand and right-hand margins of the printing are regular. (2) see left-justify and right-justify

**Kanji:** The non-phonetic Japanese writing system. In a font representing Kanji characters, each character is represented by a double-byte code. Contrast with Katakana.

**Katakana:** A character set consisting of symbols used in one of the two common Japanese phonetic alphabets. Each character is represented by one byte. Contrast with Kanji.

**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 contents) used to lead the eye horizontally. (2) The divider between text and footnotes on a page (usually a short line of dashes that can be redefined).

**left-hand page:** The page on the left when a book is opened; usually even-numbered.

**left-justify:** (1) (ISO) To control the printing positions of characters on a page so that the left-hand margins of the printing are regular.

**ligature:** A single character (piece of type or font raster) that represents two or more input characters: ff and ffi are examples of characters that may be represented by (printed as) a ligature.

**line device:** Any of a class of printers that accept one line of text from the host system at a time. SCRIPT/VS supports such line devices as the 1403 and 3800 Model 1.

**line space:** The vertical distance between the baseline of the current line and the baseline of the previous line.

**line spacing:** See *line space*.

**logical output device:** The combination of a physical output device and such logical variables as page size and number of lines per vertical inch (for line devices). A specification of 1403W6 is an example of a logical output device.

**logical page:** Synonym for page.

**lowercase:** Pertaining to small letters as distinguished from capitals; for example, a, b, g rather than A, B, G.

**machine-readable:** Data in a form such that a machine can acquire or interpret (read) it from a storage device, from a data medium, or from another source.

**maclib:** See *macro library*

**macro:** See *macro instruction*

**macro instruction:** An instruction in a source language that is to be replaced by a defined sequence of instructions in the same source language. In SCRIPT/VS, a macro is a sequence of one or more control words, symbols, and input lines. A macro's definition can be recursive.

**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 substitution:** During formatting, the substitution of control words, symbols, and text for a macro.

**map:** Associate a tag with an APF using the .AA [Associate APF] control word.

**margin:** (1) The space above, below, and on either side of the body of a page. (2) The left or right limit of a column.

**mark up:** (verb) (1) To determine the markup for a document. (2) To insert markup into a source document.

**markup:** (noun) Information added to a document that enables a person or system to process it. Markup 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 GML is a period (.).

**maximum ascender:** The maximum height from the baseline to the top of the character in the font character set.

**maximum descender:** The maximum depth from the baseline to the bottom of the character in the font character set.

**meter (m):** Basic unit of linear measurement.

**MCS:** Markup/content separator.

**millimeter (mm):** One-thousandth of a meter. There are 10 millimeters in one centimeter. (25.4 millimeters = 1 inch.)

**nonimpact printer:** A printer, such as the IBM 3800 Printing Subsystem, in which printing is not the result of mechanical impacts, but is instead produced by another process such as laser beam, ink-jet, or electro-erosion. The IBM 3800 Printing Subsystem, for example, uses a laser based technology and the 4250 Printer uses an electro-erosion process.

**object:** A sequential collection of control records that represent documents, pages, fonts, and so on.

**offset:** (1) (verb) To indent all lines of a block of text, except the first line. (2) (noun) The indention of all lines of a block of text following the first line.

**option:** Information entered with a SCRIPT command to control the execution of SCRIPT/VS.

**orientation:** The angle between the top or bottom edge of the page and the baselines within a column, measured in a clockwise direction.

**output device:** A machine used to print, display, or store the result of processing.

**output document:** A machine-readable collection of lines of text or images that have been formatted, or otherwise processed, by a document processor. The output document can be printed or it can be filed for future processing.

**output line:** A line of text produced by a formatter.

**page:** A collection of data that can be printed on a physical sheet of paper. Synonymous with logical page.

**PAGEDEF:** Synonym for page definition.

**page definition:** An object containing a set of formatting controls for printing pages of data. Includes controls for number of lines per printer form, font selection, print direction, and for mapping individual fields in the data to positions on the forms. Synonymous with PAGEDEF.

**page device:** A device that prints a formatted page that has graphics and text merged.

**page printer:** Any of a class of printers that accept composed pages, constructed of composed text and images, among other things. SCRIPT/VS supports such page printers as the 4250 Printer, the 3800 Model 3, and the 3820 Page Printer.

**page segment:** See *segment*.

**paginate:** To number pages.

**paragraph unit:** An element that has the same structure as a paragraph. In a General Document, the paragraph units are: paragraph, note, and paragraph continuation.

**parameter:** Any one of a set of properties whose values determine the characteristics or behavior of something. The syntax of some SCRIPT/VS control words includes parameters, which establish the properties of a formatting function or a printed page.

**part:** In a general document, a part is a zero-level heading segment. See also *heading segment*.

**patch PSC element:** A PSC element that is used temporarily to modify the normal output.

**pel:** The unit of horizontal measurement for the IBM 3800 Printing Subsystem and the 4250 Printer. On the IBM 3800 Printing Subsystem Model 1, one pel equals approximately 1/180th inch. On the 3800 Model 3 and the 3820 Page Printer, one pel equals approximately 1/240th inch. On the 4250 Printer, one pel equals approximately 1/600th inch.

**physical output device:** A physical device, such as a terminal, a disk file, a line printer, or a nonimpact printer. The 1403 Printer is an example of a physical output device.

**pica:** A unit of about 1/6 inch 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 character is 0.1 (1/10) inch wide. 12-pitch means 12 characters per inch, and 15-pitch means 15 characters per inch.

**point:** (1) A unit of about 1/72 of an inch used in measuring typographical material. There are twelve points to the pica. (2) In the Didot point system, a unit of 0.0148 inch. There are twelve Didot points to the cicero.

**profile:** (1) In SCRIPT/VS processing, a file that is imbedded before the primary file is processed. It can be used to control the formatting of a class of source documents. When processing GML markup, the profile usually contains the mapping from GML to APFs and the symbol settings that define the formatting style. (2) In the DLF library, a collection of information that identifies a batch SCRIPT/VS user (user profile) or a document processor (attribute profile) or that defines certain library parameters (system profile).

**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 *left-justify*.

**ragged left:** The unjustified right edge of text lines. See also *right-justify*.

**reference element:** In a general document, an element whose content is a reference to another element that is generated by an APF. There are five: figure reference, footnote reference, heading reference, index entry reference, and list item reference.

**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.

**right-justify:** (1) (ISO) To control the printing positions of characters on a page so that the right-hand margins of the printing are regular.

**row:** A horizontal arrangement of characters or other expressions on a printed page.

**rule:** (1) A straight horizontal or vertical line used, for example, to separate or border the parts of a figure or box. (2) 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 body of the page (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 body of the page (text area).

**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.

**section:** When an output page has two or more single-column parts with the same or different column-widths, or a single-column part and a multicolumn part, or two or more different multicolumn parts, each part of the output page is called a section.

**segment:** An object containing composed text and images, prepared before formatting and included in a document when it is printed.

**set:** This term is used in reference to a symbol. It implies the .SE [Set Symbol] control word.

**set size:** The set size of a given typeface determines the number of characters that will fit in a line of a given width when it is printed or set.

**small caps:** Capital letters in the same style as the normal capital letters in a font, but approximately the size of the lowercase letters.

**source document:** A machine-readable collection of lines of text or images that is used for input to a computer program.

**space:** A blank area separating words or lines.

**space unit:** A unit of measure of horizontal or vertical space. In GML markup, the em is used when a measure that is relative to the current font size is required. When an absolute measure is required, as in specifying the depth of a figure, recommended space units are inches (nnI), millimeters (nnW), picas/points (nnPnn), or Ciceros/Didot points (nnCnn), where nn is the number of units. See also *em*, *pica*, *point*, *Cicero*, and *Didot point system.*

**starter set:** An example of GML support that is provided with the Document Composition Facility. It consists of a document type description for general documents, a profile, and a library of APFs.

**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.

**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.)

**structured field:** A self-identifying string of bytes, analogous to a logical record. A structured field consists of an introducer, which identifies and characterizes the structured field, and data or parameters.

**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 can interpret the character string as a number, 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.

**string:** A linear sequence of entities such as characters or physical elements.

**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) (noun) a tab character, hexadecimal 05.

**table:** (ISO) (1) An array of data each item of which may be unambiguously identified by means of one or more arguments. (2) An arrangement of cells in rows and columns.

**tag:** In GML markup, a name for a type of document (or document element) that is entered in the source document to identify it. For example, *:p.* might be the tag used to identify each paragraph.

**terminal:** A device, usually equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

**text item:** Explicitly marked (tagged) elements that occur within text, such as within a paragraph unit. In a general document, for example, quotations and phrases are text items.

**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.

**text line:** An input line that contains only text.

**text variable:** A symbol whose final value is to be treated as text only.

**token:** A string of characters that is treated as a single entity. In SCRIPT/VS, a parameter passed to a macro in one of the local variables &*1, ... &*n.

**top margin:** On a page, the space between the body or running heading and the top edge of the page.

**translate table:** That 256-byte portion of the character arrangement table that translates the user's data code for a character recognizable by the 3800 Printing Subsystem Model 1.

**TRC:** Table reference character. In printer SYSOUT data sets, a second control byte, following the carriage control byte, which indicates which font the record is to be printed in. The presence of TRCs is indicated by the JCL parameter DCB=OPTCD=J.

**TSO:** An interactive processor within OS/VS2.

**type posture:** A typeface style variation indicating whether a typeface is upright (as in roman) or slanted to the right (as in italic or cursive).

**type size:** The vertical height (point size) of a given typeface, such as 10 point.

**type style:** Style variations in a typeface. Among these variations are posture, weight, and width.

**type weight:** The relative thickness of the strokes of a typeface. Usually described in such terms as light, demi bold, bold, and so on.

**type width:** The horizontal size (set size) of a given typeface. The width may be given in units of measurement, such as set 9 point, or it may be descriptive: ultra condensed, condensed, expanded, and so on.

**typeface:** All type of a single style. There might be several fonts (different sizes) with the same typeface or style.

**typeface family:** A collection of fonts of a common typeface that vary in size and style.

**typeset:** (1) (verb) To arrange the type on a page for printing. (2) (adjective) Pertaining to material that has been set in type.

**underscore:** (1) (noun) A line printed under a character. (2) (verb) To place a line under a character.

**unformatted mode:** (1) In document formatting, the state in which each input line is processed and printed without formatting. Other SCRIPT/VS control words remain in effect and are recognized. (2) In document printing using the UNFORMAT option, the state in which each input line (control words as well as text) is printed as it exists in the input, in the order in which it is processed. No formatting is done.

**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.)

**unit space:** The minimum amount of additional spacing acceptable for purposes of horizontal justification, as specified by the font designer.

**uppercase:** Pertaining to capital letters, as distinguished from small letters; for example, A, B, G rather than a, b, g.

**variable:** A quantity that can assume any of a given set of values.

| **variable text:** For the .VT [Variable Text] control word, text to be inserted in a formatted document by a post processor.

**vertical justification:** The process of redistributing the extra vertical white space at the end of a column in between the lines of text, so as to make each column in a set appear to be equal in depth.

**widow:** One or two lines or words at the end of a paragraph that are printed separately from the rest of the paragraph.

**word space:** The horizontal white space placed between words; referred to as an interword blank.

**word spacing:** The space between words in a line. See also *word space*.

**Writable Character Generation Module (WCGM):** A 64-position portion of the 3800 Printing Subsystem Model 1's character generation storage that holds the scan elements of one character set. There are two WCGMs in the basic 3800, and optional added storage provides two more.

# Index

| id | File | Page | Heading References |
|---|---|---|---|
| lib | DSMT0007 | vi | Publication Library Guide for the Document Composition Facility |
| intro | DSMT0008 | 1 | Introduction<br>iii |
| gml | DSMT0009 | 3 | Chapter 1. What is GML? |
| part1 | SSIMBED | 7 | Part One: Tag Guide<br>iii, 10 |
| gs | DSMT0010 | 9 | Chapter 2. Getting Started: Paragraphs and Headings<br>7, 11 |
| exer1 | DSMT0010 | 13 | Exercise: Paragraphs and Headings<br>193 |
| lst | DSMT0011 | 15 | Chapter 3. Creating Lists<br>7, 63 |
| dl | DSMT0011 | 19 | Creating Definition Lists<br>71 |
| exer2 | DSMT0011 | 26 | Exercise: Lists<br>194 |
| xf | DSMT0012 | 27 | Chapter 4. Creating Examples and Figures<br>7 |
| fgs | DSMT0012 | 28 | Figures<br>27 |
| space | DSMT0012 | 34 | Space Units<br>19, 19, 27, 30, 31, 44, 45 |
| xfxx | DSMT0012 | 38 | Exercise: Examples and Figures<br>195 |
| tables | DSMT0013 | 41 | Chapter 5. Creating Tables<br>7, 199 |
| srow | DSMT0013 | 46 | Starting a Row<br>45 |
| gorp | DSMT0013 | 55 | How to Use the ARRANGE and CWIDTHS Attributes<br>44, 44 |
| tblexer | DSMT0013 | 60 | Exercises: Tables<br>196 |
| xref | DSMT0014 | 63 | Chapter 6. Cross-References and Footnotes<br>7, 10, 17, 30, 45 |
| uref | DSMT0014 | 68 | Unmatched References |
| nohead | ? | ? | ?<br>68 |
| exer4 | DSMT0014 | 69 | Exercise: Cross-References and Footnotes<br>197 |
| exh4 | DSMT0014 | 69 | Heading Being Referred To<br>69 |
| hpq | DSMT0015 | 71 | Chapter 7. Highlighting, Citing, Noting, and Quoting<br>7, 19, 20, 20, 21, 21, 22, 42 |
| exer5 | DSMT0015 | 75 | Exercise: Highlighting, Citing, Noting, and Quoting<br>198 |
| wuf | DSMT0016 | 77 | Chapter 8. Overall Document Structure<br>7, 133 |
| fmat | DSMT0016 | 78 | The Front Matter |
| pref | DSMT0016 | 80 | The Preface |
| mifs | DSMT0016 | 84 | Combining Input Files<br>133 |
| idx | DSMT0017 | 89 | Chapter 9. Indexing<br>7, 83, 122 |
| msc | DSMT0018 | 103 | Chapter 10. Process-Specific Controls<br>7, 112, 115, 118 |

```
part2       SSIMBED      107        Part Two: Formatting Your Document
                                                   iii
sym         DSMT0019     109        Chapter 11. Symbols
                                                   106, 107
svar        DSMT0020     115        Chapter 12. Things You Can Specify At Run Time
                                                   12, 64, 107
sysv        DSMT0020     116        Instructions to the Starter Set (SYSVAR)
                                                   104, 124, 133
scmd        DSMT0020     119        Instructions to SCRIPT/VS
                                                   34, 89, 115
optio       DSMT0020     120        SCRIPT Command Options
pass        DSMT0020     120        Processing Options:
fpass       DSMT0020     121        FPASSES(n)
                                                   81, 124, 129, 199
twpass      DSMT0020     124        TWOPASS
                                                   81, 121
afont       DSMT0020     126        Font Requirements for Page Devices
                                                   121, 121
err         DSMT0021     127        Chapter 13. Common Problems—Symptoms and
                                              Solutions
                                                   81, 107, 131
sympun      DSMT0021     127        Symptom: Unexpected Formatting Results
                                                   106
symptoc     DSMT0021     128        Symptom: Page Number References Are Wrong
emsgs       DSMT0022     131        Chapter 14. Starter Set and SCRIPT/VS Messages
                                                   107, 123
xrl         DSMT0023     133        Chapter 15. The Cross-Reference Listing
                                                   68, 68, 69, 107
part3       SSIMBED      135        Part Three: Tag Reference
                                                   iii
ref1        DSMT0024     137        Chapter 16. General Rules for Markup Entry
                                                   135
asumm       DSMT0025     139        Chapter 17. Alphabetic Summary of Tags
                                                   135
sumde       DSMT0026     143        Chapter 18. Summary by Document Element
                                                   133, 135
apps        SSIMBED      147        Appendixes
                                                   iii
prim        DSMT0029     149        Appendix A. Creating and Editing a File
                                                   1, 147
vm          DSMT0029     150        Creating Documents Using VM/CMS
                                                   149
edit        DSMT0029     152        Editing Text
adding      DSMT0029     153        Adding Text
                                                   151
mvs         DSMT0029     159        Creating Documents Using MVS/TSO
                                                   149
mvsedit     DSMT0029     163        Editing Text with Line Commands
scroll      DSMT0029     163        Scrolling
                                                   161
cics        DSMT0029     169        Creating Documents in ATMS-III/CICS
                                                   149
setting     DSMT0029     169        Setting Up to Enter/Edit a Document
                                                   174, 174
create      DSMT0029     172        Creating a Document
                                                   174
store       DSMT0029     172        Storing, Replacing, and Deleting Documents
                                                   174, 178
samp2       DSMT0030     181        Appendix B. A Sample GML Document
                                                   147
solve       DSMT0031     193        Appendix C. Solutions for Exercises
                                                   147
gsx         DSMT0031     193        Markup for Getting Started—Paragraphs and
                                              Headings
                                                   13
lstx        DSMT0031     194        Markup for Lists
```

| | | | |
|---|---|---|---|
| | | | 26 |
| xfx | DSMT0031 | 195 | Markup for Examples and Figures<br>39 |
| xtb | DSMT0031 | 196 | Markup for Tables |
| 1st | DSMT0031 | 196 | First Table Exercise<br>60 |
| 2nd | DSMT0031 | 196 | Second Table Exercise<br>61, 61 |
| xrefx | DSMT0031 | 197 | Markup for Cross-References and Footnotes<br>70 |
| hpqx | DSMT0031 | 198 | Markup for Highlighting, Citing, Noting, and Quoting<br>75 |
| early | DSMT0032 | 199 | Appendix D. Differences from Earlier Releases<br>147 |
| gls | DSMT0033 | 203 | Glossary<br>21 |

## Figure ID's

| id | File | Page | Figure References | |
|---|---|---|---|---|
| xreffig | DSMT0014 | 65 | 5: | 65 |
| nofig | ? | ? | ?: | 68 |
| exf4 | DSMT0014 | 69 | 6: | 69 |
| idxf | DSMT0017 | 90 | 7: | 89 |
| ldev | DSMT0020 | 125 | 8: | 104, 124 |

## Footnote ID's

| id | File | Page | Footnote References | |
|---|---|---|---|---|
| support | DSMT0007 | vii | 1: | vi |
| adm | DSMT0007 | vii | 2: | vi, vii |
| docadm | DSMT0009 | 4 | 4: | 4 |
| perd | DSMT0010 | 9 | 5: | 9 |
| line | DSMT0010 | 10 | 6: | 10 |
| asq | DSMT0010 | 11 | 7: | 11 |
| chk1 | DSMT0010 | 13 | 8: | 13, 13, 13 |
| endtag | DSMT0011 | 16 | 9: | 16 |
| chk2 | DSMT0011 | 26 | 10: | 26, 26, 26 |
| chk3 | DSMT0012 | 38 | 11: | 38, 39, 39 |
| xxxxxx | DSMT0013 | 60 | 13: | 60, 60, 60 |
| yyy | DSMT0013 | 61 | 14: | 61, 61, 61 |
| fnxmp | DSMT0014 | 67 | 16: | 67 |
| nofoot | ? | ? | ?: | 68 |
| exer4 | DSMT0014 | 69 | 17: | 69 |
| chk4 | DSMT0014 | 70 | 18: | 70, 70, 70 |
| chk5 | DSMT0015 | 75 | 19: | 75, 75, 75 |
| files | DSMT0016 | 85 | 20: | 85 |
| chk6 | DSMT0016 | 87 | 21: | 87, 87, 87 |
| chk9 | DSMT0017 | 102 | 22: | 102, 102, 102 |
| pscrule | DSMT0019 | 112 | 23: | 112 |
| xcmd | DSMT0020 | 120 | 25: | 120 |
| atms | DSMT0020 | 120 | 26: | 120 |
| 3atms | DSMT0020 | 122 | 28: | 122, 122 |
| atms8 | DSMT0020 | 124 | 29: | 124, 124 |

| | | | | |
|---|---|---|---|---|
| atm2 | DSMT0020 | 126 | 30: | 126 |
| mono | DSMT0020 | 126 | 31: | 126 |
| monos | DSMT0020 | 126 | 32: | 126 |
| fesupp | DSMT0032 | 199 | 33: | 199 |

## Index ID's

| id | File | Page | Index References |
|---|---|---|---|
| tablt | DSMT0013 | 44 | (1) TABLE tag<br>66 |
| hi3800 | DSMT0015 | 71 | (1) 3800<br>71, 104 |
| hi4250 | DSMT0015 | 71 | (1) 4250<br>71, 104 |
| hi3820 | DSMT0015 | 71 | (1) 3820<br>71, 104 |
| hi1403 | DSMT0015 | 72 | (1) 1403<br>72 |
| sv | DSMT0020 | 116 | (1) system variables (SYSVARs) for the starter set<br>116, 116, 116, 117, 117, 118, 118, 118 |
| svo | DSMT0020 | 116 | (1) SYSVARs (system variables) for the starter set<br>116, 116, 116, 117, 117, 118, 118, 118 |
| scmmd | DSMT0020 | 119 | (1) SCRIPT command<br>120, 120, 121, 121, 122, 122, 122, 122, 122, 122, 123, 124, 124, 124, 124, 124, 124, 126, 126 |
| symps | DSMT0021 | 127 | (1) symptoms and solutions<br>127, 127, 128, 128, 128, 128, 129, 130, 130, 130 |
| api1 | DSMT0033 | 203 | (1) API |

## List Item ID's

| id | File | Page | List Item References | |
|---|---|---|---|---|
| help | DSMT0014 | 64 | '4.': | 65 |
| nolist | ? | ? | '?': | 68 |
| three | DSMT0014 | 69 | '3.': | 69 |

## Table ID's

| id | File | Page | Table References | |
|---|---|---|---|---|
| tbtest | DSMT0014 | 66 | 4: | 66 |
| notable | ? | ? | ?: | 68 |
| jb | DSMT0014 | 69 | 5: | 69 |
| alsum | DSMT0025 | 140 | 6: | |

## Imbed Trace

| | |
|---|---|
| Page 0 | (SS$TAGS H2091864) |
| Page 0 | (SPECIAL H2091864) |
| Page 0 | (GML$SYM H2091864) |
| Page 0 | (SS$M H2091864) |
| Page 0 | (SS$FM H2091864) |
| Page 0 | (SS$TITLE H2091864) |
| Page v | (LIBTBL H2091864) |
| Page xviii | (SSINTRO H2091864) |
| Page 2 | (SS$GML H2091864) |
| Page 7 | (SS$START H2091864) |
| Page 13 | (SS$LISTS H2091864) |
| Page 26 | (SS$EXFIG H2091864) |
| Page 39 | (SS$TABLE H2091864) |
| Page 61 | (SS$XREF H2091864) |
| Page 70 | (SS$HCNQ H2091864) |
| Page 75 | (SS$DOCST H2091864) |
| Page 87 | (SS$INDEX H2091864) |
| Page 102 | (SS$PSC H2091864) |
| Page 107 | (SS$SYMB H2091864) |
| Page 113 | (SS$SYSV H2091864) |
| Page 126 | (SS$PROBS H2091864) |
| Page 130 | (SS$MSGS H2091864) |
| Page 131 | (SS$XRLST H2091864) |
| Page 135 | (SS$RULES H2091864) |
| Page 138 | (SS$ALPHA H2091864) |
| Page 142 | (SS$SUMM H2091864) |
| Page 145 | (SS$MODEL H2091864) |
| Page 145 | (SS$BOXES H2091864) |
| Page 147 | (SSAPPXA H2091864) |
| Page 179 | (SSAPPXB H2091864) |
| Page 192 | (SSAPPXC H2091864) |
| Page 198 | (SSAPPXD H2091864) |

Document Composition Facility: Generalized Markup Language
Starter Set User's Guide
Order No. SH20-9186-04

**READER'S COMMENT FORM**

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. Your comments will be sent to the author's department for whatever action, if any, is deemed appropriate. Comments may be written in your own language; English is not required.

**Note:** IBM publications are not stocked at the location to which this form is addressed. Please direct requests for publications or for assistance in using your IBM system to your IBM representative or to your local IBM branch office.

| | 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? | ☐ | ☐ |
| • Do you use this publication: | | |
| As an introduction to the subject? | ☐ | ☐ |
| As a reference manual? | ☐ | ☐ |
| For advanced knowledge of the subject? | ☐ | ☐ |
| To learn about operating procedures? | ☐ | ☐ |
| As an instructor in class? | ☐ | ☐ |
| As a student in class? | ☐ | ☐ |
| • What is your occupation? | | |

**Comments:**

If you want a reply, please give your name, mailing address, and date:

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will forward your comments.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

SH20-9186-04

**Reader's Comment Form**

Fold and tape **Please Do Not Staple** Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE
International Business Machines Corporation
Department V53
P. O. Box 1900
Boulder, Colorado, U.S.A. 80301-9191

Fold and tape **Please Do Not Staple** Fold and tape

IBM

Cut or Fold along Line

Document Composition Facility: Generalized Markup Language Starter Set User's Guide Printed in U.S.A. SH20-9186-04

Document Composition Facility: Generalized Markup Language
Starter Set User's Guide
Order No. SH20-9186-04

**READER'S COMMENT FORM**

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. Your comments will be sent to the author's department for whatever action, if any, is deemed appropriate. Comments may be written in your own language; English is not required.

**Note:** IBM publications are not stocked at the location to which this form is addressed. Please direct requests for publications or for assistance in using your IBM system to your IBM representative or to your local IBM branch office.

| | 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? | ☐ | ☐ |
| • Do you use this publication: | | |
| As an introduction to the subject? | ☐ | ☐ |
| As a reference manual? | ☐ | ☐ |
| For advanced knowledge of the subject? | ☐ | ☐ |
| To learn about operating procedures? | ☐ | ☐ |
| As an instructor in class? | ☐ | ☐ |
| As a student in class? | ☐ | ☐ |
| • What is your occupation? | | |

**Comments:**

If you want a reply, please give your name, mailing address, and date:

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will forward your comments.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

SH20-9186-04

**Reader's Comment Form**

Fold and tape **Please Do Not Staple** Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE
International Business Machines Corporation
Department V53
P. O. Box 1900
Boulder, Colorado, U.S.A. 80301-9191

Fold and tape **Please Do Not Staple** Fold and tape

IBM

Cut or Fold along Line

Document Composition Facility: Generalized Markup Language Starter Set User's Guide Printed in U.S.A. SH20-9186-04

Document Composition Facility: Generalized Markup Language
Starter Set User's Guide
Order No. SH20-9186-04

**READER'S COMMENT FORM**

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. Your comments will be sent to the author's department for whatever action, if any, is deemed appropriate. Comments may be written in your own language; English is not required.

**Note:** IBM publications are not stocked at the location to which this form is addressed. Please direct requests for publications or for assistance in using your IBM system to your IBM representative or to your local IBM branch office.

| | 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? | ☐ | ☐ |
| • Do you use this publication: | | |
| As an introduction to the subject? | ☐ | ☐ |
| As a reference manual? | ☐ | ☐ |
| For advanced knowledge of the subject? | ☐ | ☐ |
| To learn about operating procedures? | ☐ | ☐ |
| As an instructor in class? | ☐ | ☐ |
| As a student in class? | ☐ | ☐ |
| • What is your occupation? | | |

**Comments:**

If you want a reply, please give your name, mailing address, and date:

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will forward your comments.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

SH20-9186-04

**Reader's Comment Form**

Fold and tape | **Please Do Not Staple** | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE
International Business Machines Corporation
Department V53
P. O. Box 1900
Boulder, Colorado, U.S.A. 80301-9191

Fold and tape | **Please Do Not Staple** | Fold and tape

IBM

Cut or Fold along Line

Document Composition Facility: Generalized Markup Language Starter Set User's Guide Printed in U.S.A. SH20-9186-04

Document Composition Facility: Generalized Markup Language Starter Set User's Guide (File No. S370-20) Printed in U.S.A. SH20-9186-04

IBM

SH20-9186-04