

Getting Started With SDML

Order Number: TBS

This book introduces SDML, the Software Documentation Markup Language, and the DOCUMENT Production System. It provides an overview of the tags used in SDML and shows how to begin creating and processing files using the system.

Revision/Update Information:	This book is a revised version of Getting Started with DOCMAP.
Operating System and Version:	VAX/VMS, Versions 3.n through 4.0
Software Version:	SDML, Version 0.9

digital equipment corporation
maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.


The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

Copyright ©1985 by Digital Equipment Corporation
All Rights Reserved • Printed in U.S.A.

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	
DECUS	RSTS	
DECwriter	RSX	

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	CANADA	INTERNATIONAL
Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario K2K 2A6 Attn: Direct Order Desk	Digital Equipment Corporation PSG Business Manager c/o Digital's local subsidiary or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215 (in Ottawa-Hull 613-234-7726).

* Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Northboro, Massachusetts 01532.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by T_EX, the typesetting system developed by Donald E. Knuth at Stanford University. T_EX is a registered trademark of the American Mathematical Society.

Contents

CHAPTER 1	QUICK START	1-1
1.1	THE FIRST RULE	1-2
1.2	THE SECOND RULE	1-4
1.3	CREATING A MEMO	1-5
1.3.1	Using SDML Tags In An Input File	1-6
1.3.2	Creating MEMO.GNC	1-7
1.3.3	Processing MEMO.GNC	1-8
CHAPTER 2	THE BEGINNER'S GUIDE TO THE SPIT BROOK PRODUCTION SYSTEM	2-1
2.1	THE TOOLS	2-1
2.2	USING AN SDML TAG: SYNTAX CONVENTIONS	2-3
2.3	CREATING HEADINGS, PARAGRAPHS, AND SIMPLE LISTS	2-4
2.3.1	Using First-, Second-, and Third-Level Headings: <HEADX>	2-5
2.3.2	Creating Paragraphs: <P>	2-6
2.3.3	Creating Lists: <ULIST>, <LE>, and <ELS>	2-8
2.4	CREATING EXAMPLES	2-10
2.4.1	Types of Informal Examples	2-11
2.4.2	Creating An Informal Code Example: <CODEEXAMPLE> and <ENDCODEEXAMPLE>	2-12
2.4.3	Creating An Informal Display Example: <DISPLAY> and <ENDDISPLAY>	2-13

Contents

CHAPTER 3	INFORMAL LISTS	3-1
<hr/>		
3.1	TYPES OF LISTS	3-1
<hr/>		
3.2	CREATING COLUMNED LISTS	3-1
3.2.1	A Two Column List _____	3-2
3.2.2	A Definition List _____	3-2
<hr/>		
3.3	FOR MORE INFORMATION	3-4
<hr/>		
CHAPTER 4	INTRODUCTION TO FORMAL EXAMPLES, TABLES, AND FIGURES	4-1
<hr/>		
4.1	CHARACTERISTICS OF A FORMAL TEXT ELEMENT	4-1
<hr/>		
4.2	THE FORMAL TAGS	4-1
<hr/>		
4.3	CREATING A SIMPLE FORMAL TABLE	4-3
<hr/>		
4.4	SUMMARY	4-4
<hr/>		
CHAPTER 5	DEFINING SYMBOLIC-NAME TAGS	5-1
<hr/>		
5.1	WHY USE SYMBOLIC-NAME TAGS?	5-1
<hr/>		
5.2	WHERE TO USE SYMBOLIC-NAME TAGS	5-2
<hr/>		
5.3	DEFINING THE SYMBOLIC NAME: THE <DEFINE> TAG	5-3
<hr/>		
5.5	TYPES OF SYMBOLIC-NAMES: LOCAL AND GLOBAL	5-4
<hr/>		
5.6	TYPES OF DEFINITION FILES: LOCAL AND GLOBAL	5-5
<hr/>		

Contents

5.7	SUMMARY	5-6
<hr/>		
CHAPTER 6	UNDERSTANDING LOGICAL STRUCTURE	6-1
<hr/>		
6.1	LOGICAL STRUCTURE VERSUS FORMAT	6-1
6.1.1	Have You Ever Used Logical Structure Before? _____	6-2
6.1.2	What Difference Does It Make, Anyway? _____	6-2
<hr/>		
6.2	HOW TO CHOOSE THE PROPER TAG	6-3
<hr/>		
APPENDIX A	QUICK REFERENCE	A-1
<hr/>		
A.1	AN OVERVIEW OF SDML TAGS	A-1
A.1.1	Structural Tags _____	A-2
A.1.2	Additional SDML Capabilities _____	A-8
<hr/>		
APPENDIX B	INPUT FILE FROM CHAPTER 1'S MEMO	b-1
<hr/>		
EXAMPLES		
1-1	The Writer's Input File _____	1-2
1-2	The Output Copy _____	1-3
2-1	Heading Levels _____	2-5
2-2	Paragraph Tags _____	2-7
2-3	List Tags _____	2-9
2-4	Informal Code Example _____	2-13
2-5	Informal Display Example _____	2-14
2-6	Informal Interactive Example _____	2-15
2-7	Informal Example Series (With Numbers) _____	2-17
2-7	Informal Example Series (Single Example) _____	2-18
2-7	Informal Example Series (With Replaced Heading) _____	2-19
3-1	A Two Column List _____	3-3
3-1	A Definition List _____	3-4
4-1	A Formal Table Example _____	4-5

Contents

FIGURES

1-3	A Memo Created From An SDML File _____	1-5
2-2	SDML Tag Syntax _____	2-3

TABLES

4-1	Tags For Formal Examples, Figures, and Tables ____	4-1
4-2	Sample Table-Formal _____	4-5
6-1	Explanations of Different Logical Structures _____	6-3

1

Quick Start

The Spit Brook Production System automates technical publishing...from the writer's manuscript to the typeset copy. You, as a writer, will find tools to present information effectively in tables and lists, tools to make necessary (and inevitable) changes, and tools to produce draft copies to circulate for review and comment.

This "Quick Start" contains exercises that let you look at and use the Spit Brook Production System. It is a hands-on approach to the system. You will:

- create an input file
- enter frequently-used SDML tags
- process your input file
- send the output file to a lineprinter or laser printer.

In other words, you will use the system at the same time you learn *how* to use it. If you already have working knowledge of Spit Brook text processing and wish to know more about the purpose of the system and the principles of development, move to Chapter 2.

Examples 1-1 and 1-2 illustrate the starting point and the end result of the Spit Brook Production System.

This example illustrates the text and tags as they appear on your screen. It is referred to as the **input file** in this chapter.

This is a hard copy illustration of a processed input file. It will be referred to as the **output copy** throughout this chapter.

Example 1-1 The Writer's Input File

```
<HEAD1>(SDML: The Software Documentation Markup Language)
<P>
Software documentation has both
<EMPHASIS>(logical structure) and physical
arrangement or formatting. Documentation shares
these characteristics with many everyday things.
For example, the cubicle you're
probably sitting in shares a logical
structure with almost every cubicle
in your building.
<P>The logical structure of a cubicle is:
<ULIST>
<LE>Terminal
<LE>Telephone
<LE>Desk
<LE>Bookcase
<LE>Wastebasket
<ELS>
<P>Now look around. How you arrange these
logical elements is the result of your
design (intentional or unintentional). You
are the "cubicle formatter" in this case.
<P>With SDML, you label the logical elements
of your document with SDML tags.
After you save the file, you call in a text formatter
to arrange the words on the page.
```

1.1 The First Rule

Take a moment to notice a few differences between the input file and the output copy.

A few of the differences are immediately obvious:

- The output copy has a heading in a different typeface.
- The output copy has paragraph indentations, bulleted lists, and other types of formatting.
- The input file has labels (called *tags*) in angle brackets (< >). They do not appear in the output copy.
- Text is entered in the input file in varying line lengths.

Example 1-2 The Output Copy

1.2 SDML: The Software Documentation Markup Language

Software documentation has both *logical structure* and physical arrangement or formatting. Documentation shares these characteristics with many everyday things. For example, the cubicle you're probably sitting in shares a logical structure with almost every cubicle in your building.

The logical structure of a cubicle is:

- Terminal
- Telephone
- Desk
- Bookcase
- Wastebasket

Now look around. How you arrange these logical elements is the result of your design (intentional or unintentional). You are the "cubicle formatter" in this case.

With SDML, you label the logical elements of your document with SDML tags. After you save the file, you call in a text formatter to arrange the words on the page.

-
- Text is not indented and spaced in the input file. It begins flush at the left margin.

By this time, you may have reached the following conclusion, the first rule of mastering SDML:

"The writer enters no formatting information into the input file."

Quick Start

Instead of entering formatting information, the writer labels the logical structure of the document using SDML tags. For more information on logical structure, see Chapter 6.

1.2 The Second Rule

Now, take another look at the two examples and observe some more subtle differences:

- Some tags are followed by text in parentheses, called an *argument*. For example, the two words *logical* and *structure* appear in parentheses after the `<EMPHASIS>` tag:

```
<EMPHASIS>(logical structure)
```

Note that *logical structure* is the only text that appears italicized in the output copy.

- Some tags are *not* followed by arguments. For example, the `<P>` tag is followed by many lines of text.

```
<P>
text text text text text text
text text text text...
```

All of these lines appear in a single paragraph in the output copy.

- In addition, there is a third condition not so readily apparent. Some tags work in teams. For example, the `<ULIST>`, `<LE>`, and `<ELS>` tags must all be present to create a list. They mark the beginning, middle, and end of a list.

These observations lead directly into the next conclusion about SDML, which is:

"Each tag has syntax, that is, conventions about how to use it correctly."

Once you know the purpose of a tag and its appropriate syntax, you're ready to use it in an input file. In the next exercise, you will create a memo using a subset of the SDML tags.

Quick Start

1.3 Creating a Memo

The short memo in Figure 1-3 was created using six different SDML tags (some more than once) entered into the input file.

Following the memo is a list of the tags used, their purpose, and their syntax.

Figure 1-3 A Memo Created From An SDML File

digital

INTEROFFICE MEMORANDUM

TO: John Smith DATE: December 13, 1984
FROM: Henry Wilson
DEPT: Publications

SUBJECT: Things You Might Not Find in a Reference Manual

Welcome to the Publications Group. As a new writer, a new employee of Digital Equipment Corp., and a new user of the VAX/VMS system, you are faced with many challenges (and 15 volumes of documentation).

However, there is a variety of sources for information around here, and I'd like to introduce you to a few:

- Documentation

Many, many more pounds waiting on-line for you to discover. Look for .MEM, .DOC, or .TYP files as a clue to readable text.

- NOTES Files

The notes files are a rich source of information for a persistent and curious mind. You can find out about software tools, business trends, state-of-the-art development, and state-of-the-mind opinions from the hundreds of contributors to the notes files. Ask your supervisor or system manager how to access them.

- Internal Publications

Some of these are public relations copy, some of them are less than formal and distributed among friends. Visit the library and/or keep your ears open.

- Experts

They're all around you. You may meet them through the network, in your group, or at a Woods meeting, but there's always somebody who knows *everything* (it seems) about whatever you're interested in.

Have fun, and if you get frustrated, remember...if everything was easy to find and understand, why would they need technical writers?

Quick Start

SDML Tag	Purpose
<DOCTYPE>	indicates the specific document type. For example <DOCTYPE>(memo) indicates the text that follows should be processed and formatted as a memo.
<TO>	names the receiver and sender of the memo. The argument lists the receiver, then a backslash (\), and the sender. You can also include the sender's phone number and mailstop on the same line, separated by slashes. Example: <TO>(receiver\sender\extension\mailstop)
<SUBJECT>	identifies the memo's subject matter. The argument contains text up to 1024 characters.
<P>	marks a new paragraph. No arguments or closing tags are needed.
<ULIST>	marks the beginning of an unordered list. Needs the <LE> tag and the <ELS> tag to complete the list structure.
<LE>	identifies a list element.
<ELS>	marks the end of a list.

1.3.1 Using SDML Tags In An Input File

Take a moment to match the tags with the formatted output.

How many <P> tags were used?

How many <LE> tags were needed?

If you cannot match all of them, look at Examples 1-1 and 1-2 at the beginning of this chapter for hints.

If you still have doubts, look at the file represented in Appendix B. It contains the input file, complete with tags, for the memo above.

After you are certain you can identify the tags used in the input file, begin the next section, which guides you through actually creating the file. Do this at your terminal, using your favorite text editor. When you finish, you will know the

Quick Start

procedure to follow to use *any* SDML tag, from simple lists and headings to figure references and tables.

1.3.2 Creating MEMO.GNC

- 1 At the DCL prompt, type:

```
edit LOGIN.COM
```

- 2 Add this line to your login file:

```
edev:[DOCTOOLS]LOCALLOGS
```

dev is the logical name for the device on your system that contains the DOCTOOLS account.

- 3 Exit and save the LOGIN.COM file.

- 4 At the DCL prompt, type:

```
oLOGIN
```

You have just executed a special command file that defines command symbols and logical names used by the Spit Brook Production System. The LOGIN.COM file executes each time you log in, so these symbols and logical names will always be available.

- 5 Create a file using your favorite text editor and name it MEMO.GNC. The extension GNC is a convention identifying a generically encoded file.
- 6 Enter the SDML tags and the text in the same fashion as they appear in Example 1-1. Do *not* separate a tag and its argument. For example,

```
<to>(Johnson\Boswell)
```

is correct, but

```
<to>  
(Johnson\Boswell)
```

or,

Quick Start

<to> (Johnson\Boswell)

is not correct, because the tag is separated from the argument.

7 Next, exit and save the file.

1.3.3 Processing MEMO.GNC

At this point, you have used one tool of the Spit Brook Production System—SDML, the Software Documentation Markup Language. You have created an input file and tagged the text as paragraphs and list elements.

Other tools translate SDML and add formatting codes to your input files. Still more tools, such as text processors, printers, and typesetters, deliver final output. As mentioned earlier, the relationship, definition, and purpose of each tool is discussed in Chapter 2. For a working knowledge of the production system, however, you need only to know which response to give to the system at various points during processing.

The next exercise produces a printed copy of MEMO.GNC.

1 At the DCL prompt, enter the following command:

```
DOCUMENT memo.gnc memo laser
```

where:

DOCUMENT	calls TEXMAC, the text formatting program.
memo.gnc	is the filename. A GNC extension is expected by default.
memo	is the document type. The <TO> and the <SUBJECT> tags are unique to the memo document type. Other doctypes and their unique tags are discussed in The SDML User's Guide.
laser	identifies the output device. Your other choices are lineprinter or typeset.

2 After you enter the DOCUMENT command and press RETURN, you should see many comments from TEXMAC that indicate the text formatting program is working. Error

Quick Start

messages may indicate that you have forgotten an angle bracket, a parenthesis, or have improperly closed a list.

- 3 When you receive the DCL prompt (\$) again, type:

```
dir memo.*;
```

You will see a number of files created while processing MEMO.GNC. The important file for laser printer output is MEMO.DVI.

- 4 Type:

```
ln01set memo
```

After you enter this command, your file is sent to the laser printer queue. Type:

```
show queue lpb0
```

Check the queue to see when your file will print.

- 5 Pick up your printed memo from the computer room.

At this point, you have used the Spit Brook Production System. If you've realized the efficiency and printed quality available, you'll be pleased to discover the options discussed in Chapter 2 and in *The SDML User's Guide*. SDML contains many more tags so you can create headings, intricate tables, or interactive and coded examples. In addition, you will learn tricks and techniques built into the system to help you avoid common problems of writing and production.

2

The Beginner's Guide to the Spit Brook Production System

The Spit Brook Production System integrates a set of tools for writers, editors, and graphic services personnel. Together, these tools increase quality and efficiency during the production process.

If you completed the "Quick Start" you've seen how the production system works. But, you may have a few questions. What were those messages about T_EX, DOCTOOLS, and TEXMAC? How do you get typeset output instead of laser printer output?

Chapter 1 gave you a glimpse of one possible document from the Spit Brook Production System, a memo. By this time you're probably thinking of other ways to make the system useful to you. This chapter describes your options. Keep in mind that the more you know about the tools, the easier you can design, build, and produce your documents.

2.1 The Tools

The following list defines the tools of the production system and describes relationships among them:

The Tool	Description
SDML	SDML is the Software Documentation Markup Language. The language consists of tags. A tag marks a logical element of a document. Organizational tags mark chapters, headings, and paragraphs, such as <CHAPTER>, <HEAD1>, and <P>. Other tags mark examples, lists, cross-references, and tables.

The Beginner's Guide to the Spit Brook Production System

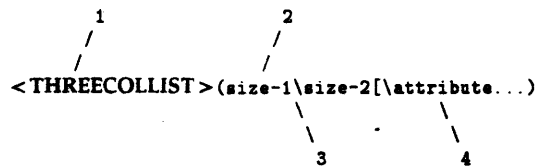
The Tool	Description
TEXMAC	A macro processor that translates a tag into commands recognizable by a text formatter. The text formatter determines the number of lines on a page, the position of chapter titles and headings, and other aspects of page layout and book design.
DSR	DSR is DIGITAL Standard Runoff, a formatting program for files destined for a lineprinter. TEXMAC can translate SDML tags into DSR.
T _E X	T _E X is a typesetting system written by Donald Knuth of Stanford University. TEXMAC can translate SDML tags into T _E X.
Lineprinter	A lineprinter produces draft copies of a document. The copies use monospaced characters without a variety of type styles and graphics. A file destined for a lineprinter is processed by TEXMAC into DSR, DIGITAL Standard Runoff.
Laser printer	A laser printer produces copy resembling typeset output. A file destined for a laser printer is processed by TEXMAC into T _E X, the typesetting system.
Typesetter	A phototypesetter produces the final paged repro ready for final art and printing.
Doctype	A doctype (document type) determines specific formatting definitions for various book designs. Familiar doctypes in the VAX/VMS group are: General Reference Manual
Template	Each document type differs in page size, margin settings, etc. When a new doctype is created, an SDML template is often available for writers. A template contains tags specific to a particular doctype. For example, the memo template has a <TO> tag and a <SUBJECT> tag unique to the memo doctype.

The Tool	Description
CMS/DOC	CMS/DOC is a library system for tracking document files as they move between writer, editor, and graphic services personnel.

2.2 Using An SDML Tag: Syntax Conventions

Figure 2-2 illustrates the general structure of an SDML tag.

Figure 2-2 SDML Tag Syntax



- 1 **Tag.** A tag can be global, template-specific, or user-defined. See Appendix A for a complete listing. Tag names can be upper or lower case.
- 2 **Argument.** Arguments can be as long as 1024 characters.
- 3 **Argument delimiter.** A backslash (\) must separate arguments.
- 4 **Keyword.** A keyword passes an option. Keywords indicate wide spacing, filled columns, etc., and are only available for a few tags.

Table 2-1 illustrates correct and incorrect usage of SDML tags. Descriptions are to the right of the tags.

Correct	Incorrect!!	Explanation
---------	-------------	-------------

The Beginner's Guide to the Spit Brook Production System

<ULIST>	<ulist	Tags must have matched angle brackets.
<DEFLIST>(10)	<DEFLIST> (10)	Argument parenthesis must immediately follow brackets. Separation by a space or a blank line is not recognized by the text formatter.
<X>(Files bold)	<X>(Files bold)	Arguments must be separated by a backslash (\)

With these syntax conventions in mind, you can tag all of the structures illustrated in the following sections.

2.3 Creating Headings, Paragraphs, and Simple Lists

Most manual pages you'll write will be divided into headings, paragraphs, and lists. Other structures, such as examples, tables, and figures, are used less frequently and will be discussed later.

This section discusses the following tags:

Tag	Purpose
<HEADX>	Labels headings. The x stands for 1, 2, or 3.
<P>	Marks a paragraph.
<ULIST>	Begins an unordered list.
<LE>	Identifies a list element.
<ELS>	Ends a list.

2.3.1 Using First-, Second-, and Third-Level Headings:
<HEADX>

Headings typographically identify a topic (usually with larger typeface or bold emphasis) and guide the reader to important information that follows. Different levels organize information into a logical hierarchy, that is:

- <HEAD1> — Most general level
- <HEAD2> — More specific topic
- <HEAD3> — Most specific topic

Here are three examples of the various heading levels:

Example 2-1 Heading Levels

Input Copy

```
<HEAD1>(This Heading is a <TAG>(Head1))  
<HEAD2>(This Heading is a <TAG>(Head2))  
<HEAD3>(This Heading is a <TAG>(Head3))
```

Output File

1.1 This Heading is a <HEAD1>

1.1.1 This Heading is a <HEAD2>

1.1.1.1 This Heading is a <HEAD3>

Notice that the output and the input have exactly the same capitalization. Headings should be capitalized according to *The Chicago Manual of Style* (be sure to consult your editor if you have questions).

Follow the coded heading with a structural tag, such as a paragraph or list. The tag that follows indicates how to format the text under each individual heading.

2.3.2 Creating Paragraphs: <P>

A <P> tag creates a new paragraph of text formatted according to the current context. For example, if the tag occurs within a list, the paragraph takes on the margins of the list. In an example or table, the paragraphs take on the format of the sections inside the example or table.

Notice the relationship of paragraphs created with <P> tags in Example 2-2.

Example 2-2 Paragraph Tags

Input Copy

```
<P>
This paragraph will have the default left margin
since it does not occur within a list.
<ULIST>
<LE>
List Item #1
<LE>
List Item #2
<LE>
List Item #3
<P>
This paragraph has the same left margin as the list
element preceding it.
<ELS>
```

Output File

This paragraph will have the default left margin
since it does not occur within a list.

- List Item #1
- List Item #2
- List Item #3

This paragraph has the same left margin as the
list element preceding it.

2.3.3 Creating Lists: <ULIST>, <LE>, and <ELS>

The <ULIST> tag marks the beginning of an unordered list. An unordered list is a series of items preceded by bullets (or by a user-defined character). Other types of lists are available in SDML:

- <NLIST> — numbered list
- <SLIST> — simple list
- <NOTEDLIST> — list with callouts
- <ALPHALIST> — list ordered by A, B, C, etc.

Each of these lists is constructed with <LE> tags and ends with an <ELS> tag. Different types of lists represent different types of information. For example, a numbered list indicated a sequence of items, an unordered list indicates that the items are merely members of a group.

is distinguished by an em dash, contains a paragraph within it, and is followed by a full-sized paragraph.

Example 2-3 List Tags

Input File

```
<ULIST>(---)
<LE>
List Item #1
<P>
The first list item in an unordered list.
<LE>
List Item #2
<P>
The second list item.
<LE>
List Item #3
<P>
The third list item.
<ELS>
<P>
Bullets precede list items when no argument
follows the <ULIST> tag.
```

Output Copy

- List Item #1
The first list item in an unordered list.
- List Item #2
The second list item.
- List Item #3
The third list item.

Bullets precede list items when no argument follows the <ULIST> tag.

2.4 Creating Examples

Most of your text will be lists or paragraphs, identified by a heading. But along with your text, you'll usually want to include many examples so your reader sees exactly what you're describing.

Just with the appearance of a page, you can show a reader that:

- an example is signified by different text.
- the example shows the difference (perhaps a different color or a different typeface) between computer output and user input.
- the example is one of a series or a single illustration of a point.

In addition, you can list the example in the table of contents with a number and caption. This is a *formal* example, and you can cross-reference it in other parts of your book. SDML constructs formal examples similarly to tables and figures. Since the concepts are so much the same, formal example tags are presented in Chapter 4 along with tables and figures. Keep in mind...any single informal example can easily become a formal example by adding (not changing) SDML tags. Chapter 4 contains examples showing an input file and an output copy.

An *informal* example is usually a short illustration of a point that needn't be listed in the table of contents, or captioned, or cross-referenced. A series of examples are also considered informal. They are handled by a special set of SDML tags that allow for intervening explanations. A series is not tagged as a formal example because a reference to a *series* of items is not very helpful to a reader.

2.4.1 Types of Informal Examples

This section discusses the following tags for creating an informal example:

Tag	Purpose
<CODEEXAMPLE>	Begins a coded example.
<ENDCODEEXAMPLE>	Ends a coded example.
<DISPLAY>	Labels screen output.
<ENDDISPLAY>	Ends screen output.
<INTERACTIVE>	Begins an interactive example.
<S>	Indicates a system prompt or response in an interactive example.
<U>	Indicates a user response or request in an interactive example.
<ENDINTERACTIVE>	Ends an interactive example.

This section also discusses the tags that label a *series* of informal examples:

Tag	Purpose
<EXAMPLES>	Begins a series of informal examples.
<ENDEXAMPLES>	Ends a series of informal examples.
<EXC>	Specifies a coded example within a series of formal or informal examples.
<EXI>	Specifies an interactive example within a series of formal or informal examples.
<S>	Indicates a system prompt or response in an interactive series.
<U>	Indicates a user response or request in an interactive series.
<EXTEXT>	Labels the explanation of the preceding coded or interactive example.

Illustrations of each of these tags follow in this section.

2.4.2 Creating An Informal Code Example: `<CODEEXAMPLE>`
and `<ENDCODEEXAMPLE>`

Code examples refer to illustrations of source code, meaning:

- the lines of a program
- the text from a generically coded file, or
- the commands from a command procedure.

Code examples are *not* illustrations of screen output, formatted text, or processed files. Those items are separate structural elements, treated in a different manner by the typesetter, and have their own SDML tags associated with them.

For Very Short Examples

You can pass a short code example as an argument (in parentheses) to the `<CODEEXAMPLE>` tag.

```
<codeexample>(OPEN/WRITE OUTFILE FILE.DAT)
```

If you surround the argument in parentheses, you do not use the `<ENDCODEEXAMPLE>` tag.

For longer examples, leave the parentheses out, enter the example exactly as you wish to see it, and end the example with `<ENDCODEEXAMPLE>`.

For Wide Examples

If you think the lines will not fit in the margins, use the optional `WIDE` argument illustrated in Example 2-4.

Example 2-4 Informal Code Example

Input File

```
<CODEEXAMPLE>(wide)
! This example must be coded "wide" because it is a very long line.
<ENDCODEEXAMPLE>
```

Output Copy

```
! This example must be coded "wide" because it is a very long line.
```

2.4.3 Creating An Informal Display Example: <DISPLAY> and <ENDDISPLAY>

Naturally, when writing a software manual, you'll want to show exactly what appears on the screen. Terminal output can be tagged with the <DISPLAY> tag. This tag gives you the freedom to enter blank lines, spaces, centered text, etc., exactly as it is seen on the terminal. For accurate results, format the display without using tab stops. Handle wide displays by using the *WIDE* argument illustrated in Example 2-4.

2.4.4 Creating An Interactive Example: <INTERACTIVE>, <S>, <U>, <ENDINTERACTIVE>

Interactive examples illustrate a dialog between the user and the system. Like the <CODEEXAMPLE> and <DISPLAY> tags, there is a beginning tag and an ending tag. The <INTERACTIVE> tag begins the example, and the <ENDINTERACTIVE> tag ends it. However, two more tags are used within the example to differentiate between the user and the system. These are:

- <S>—identifies the system prompt or response.
- <U>—identifies the user response or request.

Example 2-5 Informal Display Example

Input File

<DISPLAY>

The <TAG>(display) tag
recognizes spaces.

<ENDDISPLAY>

Output Copy

The <DISPLAY> tag
recognizes spaces.

Example 2-6 Informal Interactive Example

Input File

```
<INTERACTIVE>
<S>(Request?: ) <U>(Nameofnode)
<S>(Username: ) <U>(Yourname)
<S>(Password: ) <U>(Noecho!)
<ENDINTERACTIVE>
```

Output Copy

```
Request?:Nameofnode
Username:Yourname
Password:Noecho!
```

2.4.5 Creating A Series of Informal Examples: <EXAMPLES>, <EXC>, <EXI>, <EXTEXT>, and <ENDEXAMPLES>

Creating a series of examples using the <EXAMPLES> tag means you have a number of options:

- Numbering the example series.
- Preceding the series with a heading.
- Creating code or display examples.
- Following each example with an explanation or result.

The <EXAMPLES> tags, like the group of list tags, work together as a team. With the group of examples tags, however, your choices multiply. Once more, look at the tags that you'll use to create example series:

Tag	Purpose
------------	----------------

The Beginner's Guide to the Spit Brook Production System

<EXAMPLES >	Begins a series of informal examples.
<ENDEXAMPLES >	Ends a series of informal examples.
<EXC >	Specifies a coded example within a series of formal or informal examples.
<EXI >	Specifies an interactive example within a series of formal or informal examples.
<S >	Indicates a system prompt or response in an interactive example.
<U >	Indicates a user response or request in an interactive example.
<EXTEXT >	Labels the explanation of the preceding coded or interactive example.

The first tag, the <EXAMPLES > tag, has two optional arguments for heading information. These arguments specify numbering, special headings, or default headings:

```
<EXAMPLES >([heading-info][\heading-info])
```

The brackets around each argument indicate optional arguments. If you use the default, the examples are numbered and preceded by an "Examples" heading.

You have the following options for *[heading-info]*:

- *Example*—Suppresses the number and causes the singular heading "Example" to be output (when there is only one example).
- *Nonumber*—Suppresses the number. *Nonumber* must be passed as the second argument.
- *Nohead*—Prevents the output of a heading for the section. Examples will still be numbered.
- *Heading-text*—Causes the supplied heading text to replace the default "Examples".

The following examples illustrate three example series. The first is the default series of numbered examples preceded by the heading "Examples". The second illustrates a single "Example" with no numbers, and the third replaces the heading and suppresses the numbers.

Example 2-7 Informal Example Series (With Numbers)

Input File

```
<EXAMPLES>
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<ENDEXAMPLES>
```

Output Copy

EXAMPLES

- 1 Programming code
The function performed by this code...
 - 2 Programming code
The function performed by this code...
 - 3 Programming code
The function performed by this code...
-

Example 2-7 Informal Example Series (Single Example)

Input File

```
<EXAMPLES>(example)
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<ENDEXAMPLES>
```

Output Copy

EXAMPLE

Programming code

The function performed by this code...

Example 2-7 Informal Example Series (With Replaced Heading)

Input File

```
<EXAMPLES>(Simple Programming Examples)
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<EXC>
Programming code
<EXTEXT>
The function performed by this code...
<ENDEXAMPLES>
```

Output Copy

**SIMPLE
PROGRAMMING
EXAMPLES**

- 1** Programming code
The function performed by this code...
 - 2** Programming code
The function performed by this code...
 - 3** Programming code
The function performed by this code...
-

3

Informal Lists

3.1 Types of Lists

Some lists help the reader draw associations by grouping items in columns and rows, or by name and definition. The following tags are for definition lists, parameter lists, qualifier lists, and columned lists.

List Type	Purpose
<TWOCOLLIST>	Begins a two column list
<THREECOLLIST>	Begins a three column list
<FOURCOLLIST>	Begins a four column list
<DEFLIST>	Begins a definition list
<PARAMDEFLIST>	Begins a parameter list
<QUALDEFLIST>	Begins a qualifier list

Each list has associated tags that mark the heading, the list items, and the ending of the list.

3.2 Creating Columned Lists

SDML has tags that automatically line up columns and align numerical characters. All you have to specify is:

- the type of columned list
- the width of the column
- column headings
- column items.

The simplest columned list is two columns. A *definition list* and a *two column list* are both two columns, but their use is determined by the nature of the information.

Informal Lists

Three column, four column, and five column lists are created using the same principles as the two column list illustrated below. For more information and examples, refer to *The SDML User's Guide*.

3.2.1 A Two Column List

This section discusses the following tags for creating columned lists:

Tag	Purpose
<TWOCOLLIST >	Begins a two column list.
<TWOCOLLIST HEADS >	Labels column headings.
<TWOCOLS >	Labels column items.
<ENDTWOCOLLIST >	Ends a two column list.

Look at the following input file and output copy for an example of a two column list:

3.2.2 A Definition List

This section discusses the following tags for creating definition lists:

Tag	Purpose
<DEFLIST >	Begins a two column list.
<DEFLIST HEADS >	Labels definition list headings.
<DEFITEM >	Names the item to be defined.
<DEFDEF >	Begins the definition.
<ENDDEFLIST >	Ends a definition list.

Look at the following input file and output copy for an example of a definition list:

Example 3-1 A Two Column List

Input File

```
<TWOCOLLIST>(25)
<TWOCOLLIST_HEADS>(First Column\Second Column)
<TWOCOLS>(Item\Item)
<TWOCOLS>(Item\Item)
<TWOCOLS>(Item\Item)
<TWOCOLS>(Item\Item)
<TWOCOLS>(Item\Item)
<ENDTWOCOLLIST>
```

Output Copy

First Column	Second Column
Item	Item
Item	Item
Item	Item
Item	Item
Item	Item

Notice that the list, though identified by headings, does not have a reference number or a caption. It also will not be listed in the table of contents unless you include it as formal table. See Chapter 4 for an introduction to tags that mark a table, example, or figure as a formal text element.

Example 3-1 A Definition List

Input File

```
<DEFLIST>(10)
<DEFLIST_HEADS>(Item\Definition)
<DEFITEM>(Item #1)<DEFDEF>Definition...
<DEFITEM>(Item #2)<DEFDEF>Definition...
<DEFITEM>(Item #3)<DEFDEF>Definition...
<DEFITEM>(Item #4)<DEFDEF>Definition...
<ENDDEFLIST>
```

Output Copy

Item	Definition
Item #1	Definition...
Item #2	Definition...
Item #3	Definition...
Item #4	Definition...

3.3 For More Information

When you become proficient at creating columned lists, you'll use them quite often since they concentrate information tightly and without a lot of extra words.

You'll also find yourself demanding more sophisticated control from SDML. For example, you may find that a table extends over several pages or perhaps it is too wide for the document type.

These instructions can be passed as arguments to the tag that begins the list. For examples and correct syntax, see *The SDML User's Guide*.

4

Introduction To Formal Examples, Tables, and Figures

4.1 Characteristics of a Formal Text Element

Formal text elements were designed into the Spit Brook Production System to solve a number of documentation problems. By creating a formal example, table, or figure, you create an easy way to reference the item, include a separate file during processing, or reserve part of a page for production services after typesetting.

The most obvious features of a formal text element are:

- A unique number and label (Figure 2, Table 3, etc.)
- A caption (Figure 2-4 "Fruits and Vegetables": Graphics With The Pro 350)
- Automatic entry into the Table of Contents

4.2 The Formal Tags

In this chapter you will create a formal table, define a symbolic name for it, and set up the columns inside. You'll see that the process is very similar for a table, example, or figure.

The tags you will use are listed in Table 4-1:

Table 4-1 Tags For Formal Examples, Figures, and Tables

Tag	Description
-----	-------------

Introduction To Formal Examples, Tables, and Figures

<EXAMPLE>	Begins a formal example
<EXAMPLECAP>	Labels the example
<EXAMPLEFILE>	Includes a separate file
<NEFTPAGE>	Determines page breaks in an example
<ENDEXAMPLE>	Ends a formal example
<FIGURE>	Begins a figure
<FIGURECAP>	Labels the figure
<FIGUREFILE>	Includes a separate file
<FIGURESAPCE>	Reserves requested space for board art
<FULLPAGEFIG>	Reserves a full page
<NEFTPAGE>	Determines page breaks in a figure
<LINE_ART>	Labels a sketch drawn at the terminal
<ENDLINE_ART>	Ends <LINE_ART>
<ENDFIGURE>	Ends a figure
<TABLE>	Begins a table
<TABLECAP>	Labels the table
<TABLEFILE>	Includes a separate file
<FIRST_VALID_BREAK>	Marks the first possible break
<LAST_VALID_BREAK>	Marks the last possible break
<ALIGN_CHAR>	Used to align numbers in columns
<ENDALIGN_CHAR>	Ends <ALIGN_CHAR>
<ENDTABLE>	Ends a table

Examples for tags not illustrated later in this chapter can be found in *The SDML User's Guide*.

4.3 Creating a Simple Formal Table

The following tags are all you need to include in your file to create a formal table:

Table Tag	Description
<DEFINE>(graphics_table\1)	The <DEFINE> tag creates the symbolic name "graphics_table" and gives it a current value of 1. (See Chapter 5 for an introduction to symbolic names.)
<TABLE>(<graphics_table>)	The <TABLE> tag begins the table.
<TABLECAP>(Graphics Commands)	The <TABLECAP> tag labels the table with a caption)
<ENDTABLE>	The <ENDTABLE> tag ends the table.

Now, how does your information get sorted into columns and rows? TEXMAC, the macro processor translates the tags into a text formatter, and it is the text formatter's job to place the information correctly on the page. You need not worry about the end format when you use an SDML tag, however.

Remember, formal tags are solutions to documentation obstacles. If you have a definition list or three column list that you'd like to reference, label, and add to the table of contents, surround it with table tags. Column list, definition list, and parameter, qualifier, and keyword list structures can all be used within the <TABLE> tags.

Formal Examples and Figures

Similar to a simple table, all you need to create a formal example or figure are these tags:

Example Tags	Figure Tags
--------------	-------------

Introduction To Formal Examples, Tables, and Figures

<DEFINE>(symbolic name value)	<DEFINE>(symbolic name value)
<EXAMPLE>(symbolic name)	<FIGURE>(symbolic name)
<EXAMPLECAP>(caption)	<FIGURECAP>(caption)
<ENDEXAMPLE>	<FIGURESPACE>
	<ENDFIGURE>

All informal examples (except for example series) can be made into formal examples with the <EXAMPLE> tags.

At this stage of the Spit Brook Production System's development, all figures are placed in the document at final production.

Look at the following input file and output copy for an example of a formal table:

Notice that there are *two* ending tags: one for the table and one for the twocollist.

4.4 Summary

The formal tags create the caption, the symbolic name for cross-referencing, and the table of contents entry. Remember, however, that you still need to provide a logical text element within the formal tags. For a formal table, you can use definition lists, parameter definition lists, qualifier definition lists, and any columned list. For an example you can use a codexample, display example, or interactive example. At this stage of development, the Spit Brook Production System incorporates all art through Graphics Services, so you must leave an adequate amount of white space on the page when you want to incorporate a figure.

For more examples, see *The SDML User's Guide*.

Example 4-1 A Formal Table Example

Input File

```
<TABLE>(<SAMPLE_TABLE>)  
<TABLECAP>(Sample Table--Formal)  
<TWOCOLLIST>(25)  
<TWOCOLLIST_HEADS>(First Column\Second Column)  
<TWOCOLS>(Item\Item)  
<TWOCOLS>(Item\Item)  
<TWOCOLS>(Item\Item)  
<TWOCOLS>(Item\Item)  
<TWOCOLS>(Item\Item)  
<ENDTWOCOLLIST>  
<ENDTABLE>
```

Output Copy

Table 4-2 Sample Table-Formal

First Column	Second Column
Item	Item
Item	Item
Item	Item
Item	Item
Item	Item

5

Defining Symbolic-name Tags

5.1 Why Use Symbolic-name Tags?

Most of the time as you write documentation, developers continue to test and improve their product. A small change in a routine name or the sequence of a procedure can cause the well-known "rippling effect": not only have you written the section already, you've mentioned it, cross-referenced it, and indexed it from four other chapters in the manual. A symbolic name is a tool to make all the necessary changes automatic.

Example

In the following case, the writer created an example and gave it the symbolic name "abc_example."

```
<DEFINE>(abc_example\4)
.
.
<EXAMPLE>( <ABC_EXAMPLE>)
<EXAMPLE_CAP>(Using The ABC Command)
.
.
<ENDEXAMPLE>
```

```
<P>See Example <EXREF>( <abc_example>) for further information.
```

Defining Symbolic-name Tags

Result

The five lines in the example do the following:

- The `<DEFINE>` tag "legalizes" the tag `<ABC_EXAMPLE>` to use anywhere in the document. The current value of the tag is 4. Changing the value in the `<DEFINE>` tag changes the value of every occurrence of `<ABC_EXAMPLE>`.
- The `<EXAMPLE>` tag begins the example and names it.
- The `<EXAMPLECAP>` tag supplies the caption.
- The `<ENDEXAMPLE>` tag ends the example.
- The `<EXREF>` tag labels the reference to the example from another paragraph in the document.

When information changes, change only the definition tag, and the changes are reflected throughout the document.

5.2 Where To Use Symbolic-Name Tags

This chapter explains defining and using symbolic name-tags in your developing document. The procedure you use is essentially the same for each text element; only the individual tags differ.

Text elements that you can name and cross-reference are:

Text Element	Reference Tag
Appendixes	<code><AXREF></code> (<code><symbolic_name></code>)
Callouts	<code><CALLOUT_REF></code> (<code><symbolic_name></code>)
Chapter titles	<code><CHAPREF></code> (<code><symbolic_name></code>)
Example titles	<code><EXREF></code> (<code><symbolic_name></code>)
Figure titles	<code><FIGREF></code> (<code><symbolic_name></code>)
Glossary samples	<code><GREF></code> (<code><symbolic_name></code>)
Section titles	<code><SECREF></code> (<code><symbolic_name></code>)
Tables	<code><TABREF></code> (<code><symbolic_name></code>)

Each symbolic name must be related to an associated `<DEFINE>` tag. Using a symbolic name without a definition

Defining Symbolic-name Tags

causes an error at processing. Your symbolic name appears, with angle brackets, in your output copy.

Notice again that angle brackets enclose the symbolic-names because the <DEFINE> tag actually defines the name as an SDML tag.

5.3 Defining The Symbolic Name: The <DEFINE> Tag

The syntax for the <DEFINE> tag is:

```
<DEFINE>(symbolic-name\current value)
```

The first argument is the symbolic-name you choose for the text element; the second argument is the value you believe is current.

For example,

```
<DEFINE>(next_book\Beyond Fortran)
```

defines every occurrence of <NEXT_BOOK> to read *Beyond Fortran*.

Throughout the development of your document, you might change that line to:

```
<DEFINE>(next_book\Fortran and the Art of Motorcycle Maintenance)
```

or,

```
<DEFINE>(next_book\Fortran Koans)
```

No matter what your decision about the upcoming title, you can proceed to build all your references into the document with the symbolic name, "<next_book>".

5.4 Cross-referencing a Text Element With A Symbolic Name: <AXREF>, <BOOKREF>, <CHAPREF>, (and More)

Once you create a figure, example, or chapter, you may want to refer the reader to the text from somewhere else in the document. Use the symbolic name in place of the actual name or number to create the cross-reference.

The syntax of a cross-reference tag is:

```
<(XXX)REF>(<symbolic_name>)
```

For example, you might write:

```
See <BOOKREF>(<next_book>) for more examples, explanations, and wisdom.
```

Your currently defined value replaces the symbolic name at processing.

5.5 Types of Symbolic-names: Local and Global

A symbolic name, and therefore its definition, is *global* if referred to from another chapter or major section. It is *local* if all references reside in the chapter where it originates.

Why does this matter? When you begin writing a document, you might only have a few symbolic names, perhaps four examples, three figures, a couple of chapters. But, your document will mature. Some documents have had as many as two hundred definitions. If you divide the definitions into several local files and one global file, you have a more effective method of governing them.

Grouping the definitions or organizing them into separate files centralizes all your potential "variables" and makes updates simple.

5.6 Types of Definition Files: Local and Global

Once you understand the need for a central location of your definitions, and the difference between local and global definitions, you're ready to create the local and global definitions files.

Global Files

A global file is included only once in the document. Using a line of code in your global file from Texmac, the text processor, you can test to see if the global file has been included. Including the global file only once when you process more than one chapter makes the processing more efficient.

```
<comment>( ** Include this global file only once... ** )
<define>(end_of_file\
<ifdef>(already_included\
    |<ignore>(end_of_file)&\
    |<define>(already_included)&
<end_of_file_>
```

Comment lines label sections of your input file to help you remember or explain your code.

Local Files

Include the global file in your local file by using `<INCLUDE>`. The following line uses the logical name GLOBALDEFS for the file specification. (Logical names are widely used in the Spit Brook Production System. See *Vax/VMS Primer*, Chapter 5 for an introduction to logical names for VMS file specifications.)

```
<include>(GLOBALDEFS)
```

The remaining lines of your file will be `<COMMENT>` tags and `<DEFINE>` tags for local definitions.

5.7 Summary

Creating and maintaining definitions may seem a little confusing at first. It's an ongoing (rather circular) process throughout the development of the document, well worth the effort once you master it. Once again for review, the typical "life cycle" of a symbolic name might be like this:

Example

```
<DEFINE>(specs_tab\1)
.
.
<TABLE>( <SPECS_TABLE>\multipage)
<TABLECAP>(Hardware Specifications)
.
.
<ENDTABLE>
```

```
<P>
If your configuration matches the specifications in
table <TABREF>( <SPECS_TABLE>),...
```

Result

- The <DEFINE> tag "legalizes" the tag <SPECS_TABLE> to use anywhere in the document. The current value of the tag is 1. Changing the value in the <DEFINE> tag changes the value of every occurrence of <SPECS_TABLE>.
- The <TABLE> tag begins the table and names it.
- The <TABLECAP> tag supplies the caption.
- The <ENDTABLE> tag ends the table.
- The <TABREF> tag references the table from another paragraph in the document.

Defining Symbolic-name Tags

For More Information...

If you are just learning SDML, you're probably more concerned with getting an example, table, captioned figure, or list to process correctly than you are with processing the document as a whole. For that reason, this section only discusses defining and referencing symbolic-name tags...that is, the details you need to know to *write* your document. Managing your definitions, your major sections, and your current versions is covered in more detail in *The SDML User's Guide*. Those tasks necessary for *processing* your entire document are associated with what is sometimes called "building your book."

6

Understanding Logical Structure

This chapter explains two fundamental principles of SDML:

- Documents can be divided into their component text elements.
- Each text element can be named according to the logical function it plays within the document.

6.1 Logical Structure Versus Format

A document is made up of many component parts. For example, most documents contain:

- paragraphs
- lists
- headings
- tables
- examples.

Even finer distinctions exist in technical documentation, such as:

- definition lists
- keyword definition lists
- parameter definition lists
- qualifier definition lists.

SDML provides a name tag for each text element within a document. When you use SDML, you tag each text element with its name.

When you name a text element with an SDML tag, you name it according to its logical structure within the document. When you name a text element, you supply no information at

Understanding Logical Structure

all about how that text element should look. You simply say what that text element is.

6.1.1 Have You Ever Used Logical Structure Before?

Choosing an SDML tag to name a text element requires you to think about the logical structure of the document rather than how it might look. You choose a tag based on what the text element is rather than how it will be displayed.

If you've written an outline for a college theme, lab report, or thesis, you're already familiar with logical structure. Each of those documents ~~have~~ conventional elements, such as introductions or overviews, main body of text, various details, and a summary or conclusion. Newspapers have a familiar logical structure: headlines, text, advertising, and classified ads. (Yet, the physical format between *The New York Times* and the *Daily Enquirer* are very different.)

has

Thinking about logical structure rather than about format is something that you may find difficult at first, especially if you are familiar with controlling the format with a text formatting language.

6.1.2 What Difference Does It Make, Anyway?

After you have carefully distinguished each text element within your document, you may be puzzled to find that the formatted results of different tags are the same. In some document types, logically distinct tags may result in the same format — all *italicized*, **boldfaced**, or

in a particular style of type.

You may not be able to detect any difference in the text elements by looking at your formatted results. That's OK. You have still captured the information about logical structure by using SDML.

In another context, your book design may call for a different typographical representation for each element. For your document to be truly portable to a new or different document design, you must tag text elements according to their logical structure, independent of any formatted results.

6.2 How To Choose The Proper Tag

Choosing tags on the basis of logical structure implies that there is a clear-cut choice of tag for each text element, but at times you may be unsure about which tag is exactly right. You and your editor must decide what differentiates an argument from a parameter, for example, or a two-column list from a definition list. There are three principles to follow when you are unsure about which tag to use:

- Consult with your editor and come to an agreement.
- Use the tag consistently throughout the document.
- Use the tag consistently throughout the document set.

Table 1 explains the differences between tags for text elements that often look the same in final output. The "Description" column discusses the element's logical function, not its final appearance.

Table 6-1 Explanations of Different Logical Structures

Text Element	Tag	Description
Argument	< ARGUMENT >	Any argument passed to a routine
Key word	< KEYWORD >	A system keyword
New term	< NEWTERM >	Newly introduced term
Parameter	< PARAM >	A parameter to a procedure
Value	< VALUE >	The contents of a variable
Variable	< VARIABLE >	The name of a variable
Code Example	< CODEEXAMPLE >	The contents of a file before processing

Table 6-1 (Cont.) Explanations of Different Logical Structures

Text Element	Tag	Description
Display Example	<DISPLAY>	Output as it appears on the terminal
Interactive Example	<INTERACTIVE>	Dialog between system and user
Math Example	<MATH>	A mathematical example or expression
Syntax Example	<SYNTAX>	Notational convention
Sample Text	<SAMPLETEXT>	Text output from text processor
Definition List	<DEFLIST>	Two-column list: item and definition
Keyword Definition List	<KEYDEFLIST>	Two-column list: keyword and definition
Parameter Definition List	<PARAMDEFLIST>	Two-column list: parameter and definition
Qualifier Definition List	<QUALDEFLIST>	Two-column list: qualifier and definition

A

Quick Reference

A.1 An Overview of SDML Tags

This Quick Reference is designed for those who know the *type* of tag they want to use, but need to find the exact syntax. The following lists organize SDML tags into two classifications.

The two classifications are:

- **Structural Tags.** These tags label a specific text structure, such as paragraphs, lists, or tables.
- **Additional SDML Capabilities.** These tags allow you to work with the document, the text, or with SDML itself to manipulate the text structures or the document. For example, you use the index tag within a paragraph or list for the advantage of letting SDML create your index for you.

These two classifications are divided into twelve groups:

Structural Tags	Additional Capabilities
Headings	Emphasis
Figures	Keyboard Characters
Examples	SDML Tools
Paragraphs	Cross-References
Notes and Footnotes	Special Characters
Lists	
Tables	
Front and Back Matter	

A.1.1 Structural Tags

HEADINGS

Tag	Argument(s)
<CHAPTER>	(number title symbolic-name)
<HEAD1>	(heading-text symbolic-name GLOBAL)
<HEAD2>	(heading-text symbolic-name GLOBAL)
<HEAD3>	(heading-text symbolic-name GLOBAL)
<SUBHEAD1>	(heading-text)
<SUBHEAD2>	(heading-text)

FIGURES

Tag	Argument(s)
<FIGURE>	(symbolic-name arg-2 placement-info)
<FIGURECAP>	(caption)
<FIGUREFILE>	(logical-name)
<FIGURESPACE>	(value text)
<FULLPAGEFIG>	[(text)]
<LINE_ART>	
<ENDLINE_ART>	
<ENDFIGURE>	

Quick Reference

EXAMPLES

Tag	Argument(s)
<CODEEXAMPLE> <ENDCODEEXAMPLE>	(text)
<DISPLAY> <ENDDISPLAY>	[(WIDE)]
<INTERACTIVE> <ENDINTERACTIVE>	[(WIDE)]
<MATH> <ENDMATH>	(math-expression)
<SAMPLETEXT> <ENDSAMPLETEXT>	
<EXAMPLE> <EXAMPLECAP> <EXAMPLESPACE> <EXAMPLEFILE> <NEXTPAGE> <ENDEXAMPLE>	(symbolic-name[<i>\arg-2</i>] [<i>\placement-info</i>]) (caption) (value[<i>\text</i>]) (logical-name)
<EXAMPLES> <EXC> <EXI>	[(<i>\heading-info</i>)[<i>\heading-info-2</i>])
<S> <U> <EXTTEXT> <ENDEXAMPLES>	(text) (text) (explanation)

Quick Reference

PARAGRAPHS

Tag	Argument(s)
<P>	
<CP>	

NOTES AND FOOTNOTES

Tag	Argument(s)
<FOOTNOTE>	(callout\text)
<FOOTREF>	(callout)
<NOTE>	(heading-text)
<ENDNOTE>	
<NOTES>	[(alternate-head)]
<ENDNOTES>	

LISTS

Tag	Argument(s)
<SLIST>	
<ULIST>	[(character)]
<NOTEDLIST>	
<NLIST>	[(number)]
<LE>	[(callout-number)]
<ELS>	
<DEFLIST>	(maxtermsize[\MULTIPAGE])
<DEFLIST_HEADS>	(col-one-head-1\col-two-head-1\col-one-head-2\col-two-head-2\col-one-head-3\col-two-head-3)
<DEFITEM>	(item)

Quick Reference

Tag	Argument(s)
<DEF_ITEMS>	(item-1\def\item-2\def\item-3\def\item-4\def\item-5)
<DEFDEF>	text
<ENDDEFLIST>	
<KEYDEFLIST>	[(heading-text\NONE)]
<KEYITEM>	(item-1[\item-2\item-3\item-4\item-5\item-6])
<KEYDEF>	(text)
<ENDKEYDEFLIST>	
<KEY_SEQUENCE>	(example)
<ENDKEY_SEQUENCE>	
<NOTES>	[(alternate-head)]
<NITEM>	
<ENDNOTES>	
<PARAMDEFLIST>	[(heading-text\NONE)]
<PARAMITEM>	(item-1[\item-2\item-3\item-4\item-5\item-6])
<PARAMDEF>	text
<ENDPARAMDEFLIST>	
<QUAL_LIST>	[(arg-1\arg-2\WIDE)]
<QUAL_LIST_HEADS>	(head-1[\head-2])
<QPAIR>	(qualifier\default)
<ENDQUAL_LIST>	
<QUALDEFLIST>	[(heading-text\NONE)]
<QUALITEM>	(positive-form\negative-form [\positive-form\negative-form\positive-form\negative-form])
<QUALDEF>	text
<ENDQUALDEFLIST>	

TABLES

Tag	Argument(s)
< TABLE >	(< SYMBOLIC-NAME > [arg-2][\MULTIPAGE])
< TABLECAP >	(caption)
< TABLEFILE >	(logical-name)
< ENDTABLE >	
< FIRST_VALID_BREAK >	
< LAST_VALID_BREAK >	
< FIVECOLLIST >	(maxitemsize-1\maxitemsize-2 \maxitemsize-3\maxitemsize-4 [attribute\attribute\attribute])
< FIVECOLLIST_HEADS >	(col-one-head\col-two-head\col- three-head\col-four-head\col-five- head)
< FIVECOLS >	(item-1\item-2\item-3\item-4\item-5)
< ENDFIVECOLLIST >	
< FOURCOLLIST >	(maxitemsize-1\maxitemsize- 2\maxitemsize- 3[attribute\attribute\attribute])
< FOURCOLLIST_HEADS >	(col-one-head\col-two-head\col- three-head\col-four-head\col-one- head-2\col-two-head-2\col-three- head-2\col-four-head-2)
< FOURCOLS >	(item-1\item-2\item-3\item-4)
< ENDFOURCOLLIST >	
< THREECOLLIST >	(maxitemsize-1\maxitemsize-2\ [attribute\attribute\attribute])
< THREECOLLIST_HEADS >	(col-one-head \col-two-head\col- three-head\col-one-head-2\col-two- head-2\col-three-head-2\col-one- head-3\col-two-head-3\col-two- head-3\col-three-head-3)

Quick Reference

Tag	Argument(s)
<THREECOLS> <ENDTHREECOLLIST>	(item-1\item-2\item-3)
<TWOCOLLIST> <TWOCOLLIST_HEADS>	(maxitemsize[attribute\attribute\attribute]) (col-one-head\col-two-head\col-one-head-2\col-two-head-2\col-one-head-3\col-two-head-3)
<TWOOLS> <ENDTWOCOLLIST>	(item-1\item-2)
<ALIGN_CHARACTER> <ENDALIGN_CHARACTER>	(character)

FRONT AND BACK MATTER

Tag	Argument(s)
<FRONT_MATTER> <TITLEPAGE> <ENDTITLEPAGE> <ABSTRACT> <TITLE>	
<ORDER_NUMBER>	(number)
<SUPERSESSION>	(information)
<REVISION_INFO>	(information)
<OPSYS_VERSION>	(arg1[<i>l</i> arg2 <i>l</i> arg3 <i>l</i> arg4 <i>l</i> arg5])
<SOFTWARE_VERSION>	(number)
<COPYPAGE> <TRADEMARK_PARA> <PRINTDATE>	
<READERS_COMMENT_PARA>	(date)
<COPYDATE>	(date)
<PROPRIETARY_PARA>	

Quick Reference

Tag	Argument(s)
<CHANGE_SUMMARY>	
<ENDCOPYPAGE>	
<ENDFRONT_MATTER>	
<PREFACE>	
<ASSOCIATED_DOCUMENTS>	
<CONVENTIONS>	
<DOCUMENT_STRUCTURE>	
<INTENDED_AUDIENCE>	
<ENDPREFACE>	
<PARTPAGE>	
<RUNNING_TITLE>	(title)
<ENDPARTPAGE>	
<APPENDIX>	
<GLOSSARY>	
<GTERM>	(term)
<GDEF>	definition
<ENDGLOSSARY>	
<ENDAPPENDIX>	

A.1.2 Additional SDML Capabilities

Quick Reference

EMPHASIS

Tag	Argument(s)
<ARGUMENT>	(argument)
<BEGIN_CALLOUTS>	[(callout-number\PREFIX)]
<CALLOUT>	[(number)]
<CO>	[(number)]
<ENDCALLOUTS>	
<BITMAP>	(bits)
<EMPHASIS>	(text[\BOLD])
<KEYWORD>	(word)
<LOWERCASE>	(text)
<UPPERCASE>	(text)
<NEWTERM>	(term)
<PARAM>	(name\OPT)
<SYNTAX>	(syntax-statement)
<ENDSYNTAX>	
<TAG>	(tag-name)
<TEX>	
<UNDERLINE>	(text)
<ENDUNDERLINE>	
<VALUE>	(value-of-variable)
<ENDVALUE>	
<VARIABLE>	(variable-name)
<ENDVARIABLE>	

KEYBOARD CHARACTERS

Tag	Argument(s)
<ARROW>	(type)
<CTRL>	(letter)
<DOUBLE_QUOTE>	[(space)]
<FF>	

Quick Reference

Tag	Argument(s)
<KEYWORD>	(word)
<RET>	
<TAB>	

SDML TOOLS

Tag	Argument(s)
<COMMENT>	[(text)]
<ENDCOMMENT>	
<COND>	(condition)
<CURRENT_COND>	• Use <DEFINE> tag •
<ENDCOND>	
<DEFINE>	(tag-name\definition)
<REDEFINE>	(tag-name\definition)
<UNDEFINE>	(tag-name\definition)
<HELP_ONLY>	
<ENDHELP_ONLY>	
<INCLUDE>	(logical-name)
<LITERAL>	(literal-text)
<NOHELP>	
<ENDNOHELP>	
<X>	(index-entry\[attribute])
<Y>	(index-entry\[attribute])

CROSS-REFERENCES

Tag	Argument(s)
<AXREF>	(<SYMBOLIC-NAME>)
<BOOKREF>	(<BOOK-NAME>)
<CALLOUT_REF>	(callout)

Quick Reference

Tag	Argument(s)
<CHAPREF>	(<SYMBOLIC-NAME>)
<EXREF>	(<SYMBOLIC-NAME> [\GLOBAL])
<FIGREF>	(<SYMBOLIC-NAME> [\GLOBAL])
<GREF>	(term)
<MATHREF>	(<SYMBOLIC-NAME> [\GLOBAL])
<SECREf>	(<SYMBOLIC-NAME>)
<TABREF>	(<SYMBOLIC-NAME> [\GLOBAL])

SPECIAL CHARACTERS

Tag	Argument(s)
<AMPERSAND>	
<BACKSLASH>	
<CDB>	
<CPAREN>	
<CPOS>	(character)
<DOUBLE_QUOTE>	[(SPACE)]
<ELLIPSIS>	
<HELLIPSIS>	
<MCS>	(character)
<ODB>	
<OPAREN>	
<PARENDCHAR>	(char)
<POWEROF>	(exponent)
<QUOTE>	(text)
<ENDQUOTE>	
<SINGLE_QUOTE>	[(SPACE)]
<SUBSCRIPT>	(number)
<SUPERSCRIPt>	(number)

Quick Reference

Tag	Argument(s)
<TIMES>	
<VBAR>	

Input File From Chapter 1's Memo

<DOCTYPE>(memo)

<TO>(John Smith\Henry Wilson\Publications)

<SUBJECT>(Things You Might Not Find in a Reference Manual)

<P>

Welcome to the Publications Group. As a new writer, a new employee of Digital Equipment Corp., and a new user of the VAX/VMS system, you are faced with many challenges (and 15 volumes of documentation).

<P>

However, there is a variety of sources for information around here, and I'd like to introduce you to a few:

<ULIST>

<LE>Documentation

<P>Many, many more pounds waiting on-line for you to discover. Look for .MEM, .DOC, or .TYP files as a clue to readable text.

<LE>NOTES Files

<P>The notes files are a rich source of information for a persistent and curious mind. You can find out about software tools, business trends, state-of-the-art development, and state-of-the-mind opinions from the hundreds of contributors to the notes files. Ask your supervisor or system manager how to access them.

<LE>Internal Publications

<P>Some of these are public relations copy, some of them are less than formal and distributed among friends. Visit the library and/or keep your ears open.

<LE>Experts

<P>They're all around you. You may meet them through the network, in your group, or at a Woods meeting, but there's always somebody who knows <EMPHASIS>(everything) (it seems) about whatever you're interested in.

<ELS>

<P>

Have fun, and if you get frustrated, remember...if everything was easy to find and understand, why would they need technical writers?